

**AUTOMATIC REFERENCE MODEL DEVELOPMENT
FOR EARLY STAGE ARTIFACTS REUSE**

BY
MOJEEB AL-RHMAN AHMED SALEH AL-KHIATY

A Dissertation Presented to the
DEANSHIP OF GRADUATE STUDIES
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN, SAUDI ARABIA

1963 ١٣٨٣
In Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

In

COMPUTER SCIENCE AND ENGINEERING

February 2015

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by MOJEEB AL-RHMAN AHMED SALEH AL-KHIATY under the direction his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE AND ENGINEERING.**



Dr. MOATAZ AHMED
(Advisor)



Dr. ABDULAZIZ AL-KHORAIDLY
Department Chairman



Dr. RADWAN ABDEL-AAL
(Member)



Dr. Salam A. Zummo
Dean of Graduate Studies



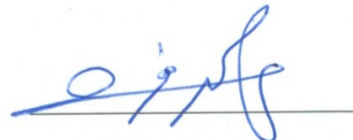
Dr. MOHAMMAD ALSHAYEB
(Member)



Dr. MAHMOUD ELISH
(Member)

6/5/15

Date



Dr. FARAG AZZEDIN
(Member)

©Mojeeb Al-Rhman Ahmed Saleh Al-Khiaty

2015

DEDICATION

*To the secret of my happiness and success,
my beloved family. |*

ACKNOWLEDGMENTS

First, and foremost, all praise and thanks are due to Almighty Allah (SWT) for giving me the health and patience to carry out this research.

I genuinely and deeply acknowledge with unlimited appreciation my thesis advisor Dr. Moataz Ahmed for guiding me along the journey of this work. Without his guidance, encouragement, support, assistance, and giant feedback, this work would not have been real. I really cannot thank him enough.

Great thanks are also due to my thesis committee members Dr. Radwan Abdel-Aal, Dr. Mohammad Alshayeb, Dr. Mahmoud Elish, and Dr. Farag Azzedin for their cooperation, giant comments, and constructive criticism.

I owe a debt of gratitude to my dear parents, the gift of Allah to me. Their constant support have made me into a better person. Words fall short in expressing my gratitude towards them. A prayer is the simplest means I can repay them - May Allah (SWT) give them good health and give me ample opportunity to be of service to them throughout my life.

I am also greatly indebted to my beloved wife and kids for their support, patience, and for bearing all the difficult times I have been facing along the course of this work.

I would also like to thank all my colleagues who supported me in many ways and who have been like a second family to me here in KFUPM.

I acknowledge the support given by Hajjah University and by King Fahd University of Petroleum and Minerals during the study of my PhD.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	VI
TABLE OF CONTENTS	VII
LIST OF TABLES.....	XII
LIST OF FIGURES.....	XIV
LIST OF ABBREVIATIONS	XVII
ABSTRACT	XVIII
ملخص الرسالة.....	XIX
CHAPTER 1 INTRODUCTION.....	1
1.1 Introduction	1
1.2 Technical Background and Motivation	4
1.3 The Research Problem	6
1.4 Research Scope	8
1.5 Research Contributions.....	8
1.6 Thesis Organization	10
CHAPTER 2 PRELIMINARIES	11
2.1 Introduction	11
2.2 Model Comparisons.....	11
2.3 Model Matching	12
2.4 Model Merging.....	13
2.5 The Unified Modeling Language (UML)	13

2.6	XMI.....	14
2.7	Reference Models	14
2.8	Genetic Algorithm (GA).....	15
2.9	Simulated Annealing Algorithm (SA).....	17
CHAPTER 3 LITERATURE REVIEW.....		18
3.1	Introduction	18
3.2	Model Consolidation: Opportunities and Challenges	19
3.3	Model Consolidation Techniques	24
3.3.1	Detecting Overlaps (Commonalities) and Differences (Variations)	25
3.3.2	Modeling Variants.....	26
3.3.3	Merging Approaches and Algorithms	27
3.3.4	Model Assumptions	29
3.3.5	Artifacts and Modeling Language Considered	29
3.4	Matching: Technical Aspects	30
3.4.1	Granularity of Matching	30
3.4.2	Identification of Similarities.....	30
3.4.3	Handling Conflicts	32
3.5	Merging : Technical Aspects	36
3.6	Summary	38
CHAPTER 4 PROPOSED FRAMEWORK.....		40
4.1	Introduction.....	40
4.2	Research Questions and Objectives	40
4.3	Illustrative Example Description	41
4.4	Solution Framework	45

4.4.1	Parsing the Input Models.....	45
4.4.2	Similarity Assessment: Model Comparison	46
4.4.3	Staged Model Matching	47
4.4.4	MSM Matrix	54
4.4.5	Filtering out Unrelated Models	55
4.4.6	Models' Merging.....	56
4.5	Summary	57
CHAPTER 5 SIMILARITY ASSESSMENT		58
5.1	Introduction.....	58
5.2	Similarity Aspects	59
5.3	Similarity Metrics	60
5.3.1	Lexical Name Similarity Metric (<i>NS</i>)	63
5.3.2	Attributes' Similarity Metric (<i>ASim</i>)	63
5.3.3	Operations' Similarity Metric (<i>OSim</i>).....	65
5.3.4	Internal Similarity Metric (<i>IS</i>).....	66
5.3.5	Neighborhood Similarity Metric (<i>NHS</i>)	66
5.4	Class-to-Class Similarity	68
5.5	Tool Support for Metrics Collection	69
5.6	Summary	69
CHAPTER 6 MODEL MATCHING.....		70
6.1	Introduction.....	70
6.2	First Stage: Element-to-Element Matching	71
6.2.1	Proposed Greedy Matching Algorithm (GGRM).....	74
6.2.2	Hybridized Greedy-Genetic Matching Algorithm (GGAM).....	76

6.2.3	Hybridized Greedy Simulated Annealing Matching Algorithm (GSAM)	84
6.2.4	Summary of the First Stage Matching.....	85
6.3	Second Stage: One-to-Many Matching.....	85
6.4	Third Stage: Residual Matching	89
6.5	Summary	91
CHAPTER 7 MODEL CONSOLIDATION.....		92
7.1	Introduction	92
7.2	Basic Concepts and Definitions.....	92
7.3	Phased Merging	97
7.4	First Phase Merging	97
7.4.1	First Pair Selection	97
7.4.2	Merging Algorithm	100
7.5	Second Phase Merging.....	106
7.6	Reference Model Properties	107
7.6.1	Reference Model Reuse.....	107
7.6.2	Reference Model Completeness.....	107
7.6.3	Reference Model Traceability and Instantiate-ability	108
7.6.4	Reference Model Reuse Recommendations.....	109
7.7	Summary	109
CHAPTER 8 EMPIRICAL ANALYSIS.....		110
8.1	Introduction.....	110
8.2	Experimental Objects	110
8.3	Accuracy Measures.....	115
8.4	Proof of Concept Tool	116

8.5	Empirical Weights Investigation	117
8.6	Empirical Investigation of Traditional Genetic versus Greedy Genetic	132
8.7	Empirical Validation of the Matching.....	141
8.8	Empirical Investigation of the Consolidation and the Ruse of the Reference Model	153
8.9	Threats to Validity	179
CHAPTER 9 CONCLUSION AND FUTURE WORK.....		182
9.1	Future work	183
REFERENCES		184
VITAE		193

LIST OF TABLES

Table 1. Matching Approaches: Similarity Information Used.....	35
Table 2. Merging Approaches.....	38
Table 3. Handling the Research Questions Throughout the Thesis' Chapters	41
Table 4. <i>ES</i> Matrix, Pair-wise Similarity Between M_0 's Classes and M_2 's Classes	50
Table 5. MSM Matrix, the Matched Classes' Similarity	51
Table 6. Second Stage Similarity Matrix (GS Matrix)	52
Table 7. MSM Matrix, the Matched Classes Similarity After 2 nd Stage Matching	54
Table 8. MSM Matrix, the Matched Classes Similarity After 3 rd Stage Matching	54
Table 9. Pearson Correlation Between Path Length and WuP Semantic Similarity Measures	63
Table 10. Lookup Table of Similarities between Relationships' Ends in Class Diagram.....	67
Table 11. Weight Settings of the Compound Metrics.....	68
Table 12. GA Parameters Settings	83
Table 13. Pair-wise MSM Matrices of Models M_0 , M_1 , M_2 , and M_3	95
Table 14. Reference Model Preliminary Catalog (RMPC).....	105
Table 15. Perturbation Performed by the Instance Generator.....	112
Table 16. Basic Statistics about the Case Studies.....	113
Table 17. Empirical Investigation Roadmap	114
Table 18. The Accuracy Obtained at Some Special Cases of the Weight Assignment, <i>Nsim</i> Metric, Equation (11).....	120
Table 19. The Accuracy Obtained at Some Special Cases of the Weight Assignment for the Metrics <i>NIS</i> , <i>NNHS</i> , <i>INHS</i> , <i>NINHS</i> , Equation (12) through (15), Case Study 1	126
Table 20. The Accuracy Obtained at Some Special Cases of the Weight Assignment for the Metrics <i>NIS</i> , <i>NNHS</i> , <i>INHS</i> , <i>NINHS</i> , Equation (12) through (15), Case Study 2	128
Table 21. The Accuracy Obtained at Some Special Cases of the Weight Assignment for the Metrics <i>NIS</i> , <i>NNHS</i> , <i>INHS</i> , <i>NINHS</i> , Equation (12) through (15), Case Study 3.	130
Table 22. Low perturbation, Case Study 3	131
Table 23. Best Value Achieved by Traditional GA at Different Problem Sizes	139
Table 24. Time Taken by Traditional GA and GGAM over 20000 Iterations at Different Problem Sizes	139
Table 25. Matching Time Taken by GGRM, GGAM, and GSAM Algorithms for Each Pair of Models.....	151
Table 26. Pair-wise Models' Similarity between 6 Input Models	155
Table 27. The Average Similarity of Each Model to the Other Models.....	155
Table 28. Pair-wise Models' Similarity After Removing M_4	155

Table 29. Pair-wise Models' Similarity After Removing M_4 and M_5	155
Table 30. Optional Point Commonality, Case Study 3	168
Table 31. Reference Model Commonality, Case Study 3.....	168
Table 32. Optional Point Commonality, Case Study 1	169
Table 33. Reference Model Commonality, Case Study 1	169
Table 34. The Standard Deviation of the Reuse Ratio over the Different Runs.....	172
Table 35. Optional Points Ratio Overhead at Different Size of the Reference Model, Case Study 3	175
Table 36. Optional Points Ratio Overhead at Different Size of the Reference Model, Case Study 1.	175

LIST OF FIGURES

Figure 1. Input Models Representing Four Different Applications of Flight System	42
Figure 2. Reference Model for the Simple Flight Booking System	44
Figure 3. Reference Model Consolidation Framework.....	46
Figure 4. Steps of the Second Stage Greedy Matching Algorithm.....	53
Figure 5. Data Type Taxonomy	66
Figure 6. SGRM Algorithm	73
Figure 7. Pair-wise Element Similarity Matrix Between Classes of Two Models, M1 and M2	73
Figure 8. Matching Similarity Matrix between Classes of Two Models, SGRM Algorithm.....	73
Figure 9. GGRM Algorithm	74
Figure 10. An Illustrative Example of the Proposed Greedy Matching Algorithm (GGRM)	75
Figure 11. Matching Similarity Matrix between Classes of Two Models, GGRM Algorithm.....	76
Figure 12. Genetic Crossover	83
Figure 13. Second Stage Matching Algorithm	86
Figure 14. An Illustrative Example of the Steps of the Third Stage Matching Algorithm.....	90
Figure 15. Selection of the First Pair for Merging.....	99
Figure 16. First Phase Merging Steps	102
Figure 17. Trace Matrix Showing Classes' Distribution over Different Instances, Case Study 3.	113
Figure 18. Pseudo Code of the Weight Calibration of the Constituents of <i>NSim</i> Metric, Equation (11).....	118
Figure 19. Models' Matching Accuracy at Different Weight Settings for the Neighborhoods Similarity Metric <i>NHS</i> ' Constituents, Equation (11).	119
Figure 20. Models' Matching Accuracy at Different Weight Settings for the <i>NINHS</i> Similarity Metric Constituents, Equation 12 through 15, Case Study 1	123
Figure 21. Pseudo Code of the Weight Calibration of the Constituents of <i>NINHS</i> Metric, Equation (15).....	124
Figure 22. Models' Matching Accuracy at Different Weight Settings for the <i>NINHS</i> Similarity Metric Constituents, Equation 12 through 15, Case Study 2.....	127
Figure 23. Models' Matching Accuracy at Different Weight Settings for the <i>NINHS</i> Similarity Metric Constituents, Equation 12 through 15, Case Study 3	129
Figure 24. Models' Matching Accuracy at Different Weight Settings for the <i>NINHS</i> Similarity Metric Constituents, Equation 15, Case Study 3 (Low Perturbation)	132
Figure 25. Synthetic Data Generator for ES Matrix	134

Figure 26. Example of the Simulated ES Matrix Generated by the Synthetic Data Generator.....	136
Figure 27. Traditional GA versus GGAM, the Convergence of the Fitness Function to the Optimal Value.....	138
Figure 28. The Convergence of Hybridized GA versus Traditional GA in the First 200 Iterations, n=50	139
Figure 29. GA versus GGAM Algorithm, Matching Accuracy, Precision and Recall, Case Study 1	140
Figure 30. Convergence Behavior of Traditional GA versus GGAM over the First 200 Iterations	141
Figure 31. Matching Accuracy, Precision and Recall of GGRM, GGAM, and GSAM, Case Study 1	143
Figure 32. Matching Accuracy, Precision and Recall of GGRM, GGAM, and GSAM, Case Study 2.....	144
Figure 33. Matching Accuracy of GGRM, GGAM, and GSAM, Case Study 3	152
Figure 34. Snapshots from the Reference Model Catalog, Case Study 0.....	157
Figure 35. Snapshots from the Reference Model Catalog, Case Study 2.....	158
Figure 36. Reference Common Classes Evolution as More Instances Are Added to the Reference, Case Study 3	160
Figure 37. Reference Common Classes Evolution as More Instances Added to the Reference, Case Study 1	161
Figure 38. Optional Points Creation and Reuse During Generalization, Case Study 3..	163
Figure 39. Optional Points Creation During Generalization, First Pair, Case Study 3...	164
Figure 40. Optional Points versus Number of Instances in the Reference Model, Case Study 3	164
Figure 41. Optional Points versus Number of Instances in the Reference Model, Case Study 1	164
Figure 42. Number of Optional Points Added Due to the Generalization of a New Instance versus the Size of the Reference Model, Case Study 3	165
Figure 43. Number of Optional Points added Due to the Generalization of a New Instance versus the Size of the Reference Model, Case Study 1	165
Figure 44. Percentage of Single Instance Optional Points against the Number of Instances in the Reference Model, Case Study 3	166
Figure 45. Percentage of Single Instance Optional Points against the Number of Instances in the Reference Model, Case Study 1	166
Figure 46. Percentage of Multiple Instances Optional Points against the Number of Instances in the Reference Model, Case Study 3	167
Figure 47. Percentage of Multiple Instances Optional Points against the Number of Instances in the Reference Model, Case Study 1	167
Figure 48. Percentage of Optional Points at Different Commonality Values, CS3.....	169

Figure 49. Percentage of Optional Points at Different Commonality Values, Case Study 1	170
Figure 50. Average Reuse Ratio in a New Instance versus the Size of the Reference Model, Case Study 3.....	172
Figure 51. Average Reuse Ratio in a New Instance versus the Size of the Reference Model, Case Study 1.....	172
Figure 52. Attributes Added to the Reference Class Per New Instance, CS3.	173
Figure 53. Methods Added to the Reference Class Per New Instance, Case Study 3	174
Figure 54. Number of Optional Points Per Instance versus Reference Size, Case Study 3	176
Figure 55. Number of Optional Points Per Instance versus Reference Size, Case Study 1	176
Figure 56. The Ratio of Optional Points to the Optional Classes versus the Reference Model Size, Case Study 3	177
Figure 57. The Ratio of Optional Points to the Optional Classes versus the Reference Model Size, Case Study 1	177
Figure 58. The Ratio of Optional Points to the All Classes versus the Reference Model Size, Case Study 3	177
Figure 59. The Ratio of Optional Points to the All Classes versus the Reference Model Size, Case Study 1	178

LIST OF ABBREVIATIONS

GA	: Genetic Algorithm
SGRM	: Simple Greedy Matching Algorithm
GGAM	: Greedy-Genetic Matching Algorithm
GGRM	: Global Greedy Matching Algorithm
GSAM	: Greedy-Simulated-Annealing Matching Algorithm
IS	: Internal Similarity
INHS	: Internal and Neighborhood Similarity
NHS	: Neighborhood Similarity
NINHS	: Name, Internal and Neighborhood Similarity
NIS	: Name and Internal Similarity
NNHS	: Name and Neighborhood Similarity
NS	: Name Similarity
RM	: Reference Model
RM	: Reference Model Catalog
RMPC	: Reference Model Preliminary Catalog.
UML	:Unified Modeling Language
XMI	:XML Metadata Interchange

|

ABSTRACT

Full Name : [MOJEEB AL-RHMAN AHMED SALEH AL-KHIATY]
Thesis Title : [Automatic Reference Model Development for Early Stage Artifacts Reuse]
Major Field : [Computer Science and Engineering]
Date of Degree : [February 2015]

Software reuse has been regarded as the key strategy for overcoming the software crisis. Reuse has great potential when systematically planned and managed to capitalize on the commonalities that exist among the different applications within the same or similar domains. Additionally, reuse of early-stage artifacts has great potential as compared to later-stage artifacts reuse. However, using multiple models to achieve the reuse potential across them is impractical and complex, especially, when models are of large size.

Early-stage reference models have been considered as good tools to allow reuse across applications within the same domain. They can offer the reuse potential of the models they consolidate and represent with manageable complexity. However, there has not been enough research to address the problem of automatically consolidating a given set of analysis (design) models representing different applications (instances) in a domain into a reference model that represents the input models.

This thesis addresses this problem and offers an approach consisting of staged matching and merging algorithms to identify commonalities and variabilities among input models, and proposes a reference model accordingly. Our focus in this thesis is on the structural models represented by class diagrams. We compared different heuristic algorithms including genetic algorithms and simulated annealing in dealing with the complexity of the matching and merging problems. We conducted a set of experiments using a number of case studies. The experiments show that our approach is promising.

ملخص الرسالة

الاسم الكامل: مجيب الرحمن أحمد صالح الخياطي

عنوان الرسالة: البناء التلقائي للنماذج المرجعية كوسيلة لتدعيم اعادة الاستخدام لمكونات الانظمة البرمجية في مراحلها الاولى.

التخصص: علوم وهندسة الحاسب الآلي

تاريخ الدرجة العلمية: فبراير 2015

ان بناء برمجيات جديدة من خلال اعادة استخدام مكونات برمجية موجودة مسبقاً تعتبر استراتيجية اساسية للتغلب على ازمة البرمجيات والمتمثلة في الكلفة والوقت والكفاءة. تكمن فائدة اعادة الاستخدام بشكل افضل عندما يتم التخطيط لها وادارتها بشكل نظامي ضمن نطاق معين او نطاقات متشابهة حيث تشترك العديد من البرمجيات ضمن النطاق الواحد او النطاقات المتشابهة في الكثير من الوظائف. اضافة الى ذلك فإن اعادة الاستخدام للمكونات البرمجية في المراحل الاولى من دورة حياة تطوير البرنامج تكون ذات فائدة جمة اذا ما قارناها بالفائدة المرجوة من اعادة استخدام المكونات البرمجية في المراحل المتقدمة من دورة حياة البرمجيات. بالرغم من هذه الفائدة المرجوة الا ان تحقيقها بشكل فعلي من عدة نماذج ليس بالشيء السهل، خاصةً عندما تكون النماذج ذات احجام كبيرة.

تعتبر النماذج المرجعية لمكونات البرمجيات في مراحلها الاولى اداة جيدة لتدعيم اعادة الاستخدام من عدة انظمة برمجية في نطاق ما، حيث يستطيع النموذج المرجعي تقديم نسبة اعادة الاستخدام الكامنة في عدة نماذج وذلك من خلال نموذج واحد تسهل ادارته، حيث يشمل هذا النموذج المرجعي العناصر المشتركة والمختلفة بين تلك النماذج.

بالرغم من اهمية تلك النماذج المرجعية الا انها لم تُعط الاهتمام الكافي من قبل الباحثين من حيث كيفية بنائها بشكل اوتوماتيكي من مجموعة من النماذج المنفردة.

يقدم هذا البحث حلاً متضمناً خوارزميات مرحلية لكلٍ من مطابقة النماذج المختلفة و دمجها الى نموذج مرجعي واحد. محور تركيزنا في هذا البحث هو نماذج البنية الهيكلية لانظمة البرمجيات ممثلة بنماذج الفئة. حيث قمنا بمقارنة عدة خوارزميات بما فيها الخوارزميات الجينية وخوارزميات محاكاة تبريد الصلب، من اجل التعامل مع تعقيدات مسألتي المطابقة والدمج للنماذج والتغلب عليها. لقد تبين من خلال التجارب العملية التي قمنا باجراءها على حالات دراسة متعددة ان الحل المقترح ذو جدوى.

CHAPTER 1

INTRODUCTION

1.1 Introduction

The ability to ship a new software product with high quality, within a short timeframe, and with sustainable profit has been vital for software companies to keep up with the new business opportunities [1-3]. These three important aspects of the software development have been coined within the software engineering community as the *software crisis*, which is the main motivation for the adoption of the engineering approach, in the late 1960s, to the software development to make it an engineering discipline [4].

Mature engineering disciplines have several handbooks that describe successful solutions to known problems. This wealth of knowledge is the accumulative contributions of dozens of top experts in the field. If software engineering is to become a mature engineering discipline, successful practices must be systematically documented so that it can be widely disseminated and reused [5].

Software reuse has been regarded as the key strategy for overcoming the software crisis [4, 6-8]. It is the process of building new software systems by the use of engineering knowledge or artifacts from existing systems rather than building software systems from scratch [4, 8, 9]. As software engineering is becoming a mature engineering discipline, successful practices must be systematically documented so that they can be widely disseminated and reused [5]. *Systematic software reuse* is an effective way to significantly improve software development [7, 9]. It reduces the risk of development

errors, leverages existing resources, transfers knowledge and experience from experts to the novice, leads to reductions in software development cost and time, and promotes high quality software. Additionally, reuse has great potential when systematically planned and managed in the context of a specific domain, where application families share some functionality [10, 11]. This common functionality, if managed appropriately, is the actual reuse pay-off and the crucial factor to the reuse success [11-13]. Thus, the *goal* of researchers with regard to software reuse is to come up with systematic procedures for engineering new systems from existing assets [9, 13-15].

The notion of reuse is not new in the software development domain. Software engineers have been reusing algorithms, code and other artifacts for long time. Hence the problem is not the lack of software reuse, rather, it is the way the artifacts have been being reused. Traditional software reuse practices are ad hoc [16], even at the model level [17]. Under the pressure of constantly changing requirements entailed by the dynamic business world, engineers are driven by the opportunistic thought of copy and modify reuse [18], and thus, inevitably, find themselves dealing with large collections of models. These models represent different versions across time, different applications in a domain, different development concerns and so on [19]. Additionally, these models represent a main source of knowledge which is captured from the minds of people involved. This knowledge is re-practiced each time new software is created, yet, when comparing software systems, we usually find 60% to 70% of a software product's functionality is common [20]. Thus, without effective reuse mechanism, it is possible to build a new system from scratch, yet a similar situation has been built before. This results in redundant artifacts, and thus redundant maintenance cost and time for the duplicated

artifacts. Thus, it is very much needed to have a systematic way to access and reuse existing software models in an efficient way.

One approach with a great potential here is to consolidate these models into a single model that unifies their commonality and explicate their variabilities. We require that such single model must have the following properties [19]:

- It offers the reuse potential of the set of models it generalizes while keeping the complexity at level of a single, yet more complicated*, model.
- *Completeness*: The model must be *complete* in the sense that if an element appears in one of the source models, it must be represented in the merged model as well.
- *Minimum redundancy*: Identical elements appearing in more than one instance must be unified into a single element in the consolidated model.
- *Traceability*: each element in the reference model is traceable to its original instance;
- *Instantiate-ability*: each input instance can be instantiated back from the reference model.
- *Information Representation*: the representation of the reference model is informative enough in such a way that it can guide the reuser about the common analysis and design practices in the domain.

* The complexity comes from the need to handle and represent the variability among the different instances.

1.2 Technical Background and Motivation

As mentioned earlier, the notion of reuse is not new in the software development domain. Software engineers have been reusing algorithms, data structures, and code blocks (routines, components, libraries) since programming was started. The recognized benefits of code reuse have encouraged its practice across the entire software development life-cycle, starting with *domain modeling* through *requirements specification*, *software design*, *coding* and *testing*, to *maintenance* and *operation* [21]. We refer to the first three types of artifacts (*domain modeling*, *requirements specification*, and *software design*) as *early-stage* reusable artifacts while the rest are referred to as *later-stage* reusable artifacts. Reuse at the level of early-stage artifacts has been acknowledged to be more beneficial than reuse of later-stage artifacts [21, 22]. This is due to the fact that in early-stage reuse, once a match is found, all related later stages artifacts for the match can also be reused [22]. Additionally, the benefit of code level reuse is limited due to the fact that the underlying software technology is moving so fast, especially true in software projects with long time scales [23]. Moreover, it is generally known that coding represents not more than 25% of the cost of system development [16].

Shifting the engineering focus during system development from late-stage artifacts (i.e. code) to early-stage artifacts (i.e. models) is the aim of *Model-Driven Development* [24] (MDD) — a software development methodology which emphasizes the use of models as the primary artifacts in the development process [25]. This implies that software developers working within this paradigm should be able to automatically generate software systems directly from models, without going through the step of writing computer code (text-based). Thus, the goal of MDD is to migrate from a code-

centric approach towards a model-centric approach, thereby separating business logic from implementation details and getting domain experts more directly involved in the development process [26]. The level of abstraction, provided by MDD, per se, saves substantial time and resources in production and delivery through: identifying and resolving defects/errors early and thus reducing rework; downscaling the complexity underlying software systems' requirements, easing communication between stakeholders, and reusing the early stages artifacts and knowledge in the subsequent stage (construction) through an automated process [21, 22, 27].

As mentioned earlier, traditionally, software artifacts' reuse along the software development life cycle has been driven by the copy-and-modify thought. Object-oriented design patterns [28] have been one of the most significant and successful ideas in software developments that support the systematic reuse at the design level. They are the vehicles that transfer design knowledge and experience from experts to the novice. One of the basic goals of design patterns is to capture already proven and matured design solutions, in the form of co-operating classes, so that addressing specific recurring design problems does not always have to start from scratch. However, design patterns target small-grained reuse, i.e. reuse at the micro-architecture level.

Software Product Line (SPL) is an emerging methodology that systematically supports early stage artifacts' reuse. It offers a strategic and promising approach for architecture reuse (i.e. coarse-grained reuse) within a family of products [29]. It provides an efficient mechanism for managing the commonalities and variabilities among a family of products. Modeling commonalities and variabilities is a key concept in development

for reuse. SPL commonalities refer to artifacts that are part of each product of the product line, whereas the SPL variabilities refer to artifacts that are specific to some products.

Synergizing the abstraction capability provided by the MDD with the variability management capability of SPL engineering bears the potential benefits of both [30, 31]. However, unless we have enough understanding and experience of the market needs about the underlying domain (or a similar domain), it is difficult to foresee what is common and what is variable among a family of software products upfront, and thus it becomes difficult, skeptical, and risky for the software development company to follow the traditional (proactive) software product line approach [18, 32, 33]. Due to the aforementioned issues, proactive product line approach (i.e. SPL first) is rarely used, and usually dominated by reactive (i.e. extending existing SPL) or extractive (i.e. building new SPL from multiple products) approaches. Therefore, when there is a collection of similar software development artifacts the extractive (also called bottom-up) approach is the most applicable to integrate these artifacts in a way that provide an efficient and effective reuse environment [32].

1.3 The Research Problem

Software reuse has been regarded as a key to overcome the software crisis [4, 6-8]. Reuse of early-stage artifacts has a great potential as compared to later-stage artifacts reuse [21, 22]. Additionally, reuse has great potential when systematically planned and managed in the context of a specific domain, where application families share some functionality. The theoretical reuse potential within the same domain can be up to 85% [34, 35]. This reuse potential capitalizes on the commonalities that exist among the different applications within the same or similar domains [12]. However, dealing with

multiple models to achieve the reuse potential across them is impractical and complex, especially, when models are of large size. Reference models have been considered as good tools for generalizing the domain practices, by capturing their commonalities and differences, to allow reuse across applications within the same domain or across similar domains.

Despite the considerable efforts that have been made by researchers towards building a generic artifact out of a set of existing ones [18, 19, 33, 36-43], some notable challenges still exist concerning the following building blocks of such a generalization process: 1) the development of a *solid similarity assessment mechanism* that uses efficient *comparison algorithm* and *matching algorithm* along with accurate *similarity measures* for comparing the different artifacts and identifying their commonalities and variabilities at different levels of granularity; 2) the development of an efficient *consolidation mechanism* along with efficient algorithms to generalize the elements of the input models into the reference model so that their commonalities are unified and variabilities are explicated at different levels of granularity; 3) a scheme for representing the different level of similarity between the input instances as an interface for bridging the gap between the output from the matching algorithms and the input to the merging algorithms bearing in mind that the software product line is thought of as background blue-prints; 4) a scheme for *reference models* representation that preserves the necessary information needed for tracing artifacts of a given reference model to their corresponding instances and vice versa; 5) providing a tool support to automate the consolidation process throughout all of its different stages.

Addressing the above mentioned challenges is expected to increase the opportunities of early stages reuse, improve the developer productivity, guide the large-scale early stages reuse of the software development artifacts, reduce maintenance cost, reduce rework, and result in high quality product.

1.4 Research Scope

Typically, for each software system, there is a set of models that describe its structural, behavioral, and functional perspectives. We focus in this work on the structural perspective, modeled by the UML (Unified Modeling Language) class diagram. Therefore, the word ‘model’ henceforth will refer to a UML class diagram at both the analysis and design stages of software development.

UML class diagram is the most important static representation in object oriented software projects [44]. It is the diagram that models the real world objects and the relationships among them. It is also the diagram that model-to-code transformation tools use first and foremost [44, 45].

1.5 Research Contributions

The main contributions of this thesis work are as follows:

- Conducting an extensive critical survey of: the existing approaches that have been addressing the problem of consolidating a set of existing models to build a single reference model; the information considered to assess the similarity and differences between such models; the requirements that should be considered by the comparison or merging algorithms or tools; fundamental challenges involved in such a consolidation process.

- Proposing a staged consolidation framework for generalizing a set of analysis (design) instances representing different applications in the domain into a reference model. Within this framework we propose:
 - staged comparison and matching algorithms for identifying the commonality and variability among the set of instances to be generalized.
 - a well-defined interface that define the output of matching algorithm and the input to merging algorithms in terms of different similarity levels so that the complexity of the consolidation problem can be broken down.
 - staged merging algorithms for handling the commonality and the variability among the set of instances to be generalized and at different level of granularity.
- Proposing a representation mechanism for:
 - representing the common, the variants, and the optional elements, among the input models, in the reference model.
 - allowing the elements in the reference model to be traced back to their original instances.
 - enabling the instantiation of original instances from the reference model.
 - guiding the reuser about the most common practices in the domain.
- Developing a proof-of-concept Java-based tool for implementing the consolidation framework with the following main properties:
 - computing different similarity metrics with configurable weight settings.
 - providing an implementation for five model matching algorithms.
 - providing two XMI parser for two modeling tools: Altova and ArgoUML

- providing an implementation for the proposed merging algorithms

1.6 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 provides technical background. Chapter 3 summarizes the related work. The conceptual description of the solution framework, through an illustrative example, is presented in Chapter 4. Chapter 5 discusses model comparison and different similarity aspects. This is followed by the staged model matching in Chapter 6. The staged merging is detailed in Chapter 7. Empirical investigation of the proposed solution is discussed in Chapter 8. The thesis findings and future directions are summarized in Chapter 9.

CHAPTER 2

PRELIMINARIES

2.1 Introduction

Models in software development allow engineers to downscale the complexity of the software systems [46]. They are the developer means for reasoning about the requirements, communicating with stakeholders, documenting the system to ease the maintenance task, generating test cases, etc [42, 47].

As models have been promoted to primary artifacts in software development, an efficient model management becomes a necessity [48, 49]. Global model management operations involve, among others, model comparisons, model matching, and model merging (also known as model consolidation) [50, 51].

Both model comparison and model matching are at the core of different model management operations such as model evolution [43], consolidation [41], and retrieval. An accurate identification of the similarity and differences between the elements of the matched models leads to an accurate model matching, which, in turn, leads to better model management.

The rest of this chapter provides background on the different concepts, techniques and technologies used in this dissertation.

2.2 Model Comparisons

Model comparison is the task of assessing or quantifying the degree of the similarity between the elements of the compared models [52, 53]. Crucial to an efficient similarity

assessment is to have a set of similarity metrics that considers the various aspects of the compared models, thus their overlaps and differences are best quantified. Software metrics are the software engineer means to quantify the similarity between the elements of the compared models. In the context of the class diagram, a metric which measures the similarity between two classes based on their names similarity is an example of such metrics.

2.3 Model Matching

Model matching is the task of determining the correspondence between the elements of the compared models [19, 54-57]. Within the context of our thesis we define model matching between a pair of two models as the task of mapping each element in the smaller model of the pair (model with fewer number of classes) into its most similar element in the other model, given the similarity scores between the elements of the two models as quantified by the similarity metrics. Accurate similarity assessment (comparison) leads to accurate matching, and accurate matching leads to a duplication-free merging [58].

Model matching task is time consuming due to the fact that finding the optimal match between the elements of two models is a kind of combinatorial problem generally referred to as graph matching problem [59]. Therefore, an efficient matching algorithm is required to obviate the complexity of the brute-force method and meanwhile provide an acceptable solution. One of the approaches is to make some plausible assumptions which can be driven by utilizing the characteristics of the problem in hand. Alternative way is to go with some heuristic based solutions, e.g. Genetic Algorithms.

Model matching techniques can be classified into exact model matching [43, 60] and inexact model matching [33, 50, 61]. The exact model matching aims at finding a strict correspondence between the two models to be matched, or between sub-sets of their elements. This makes it so restrictive and impractical to the expected variations that may exist among the elements of the matched models, and thus it usually fails to find feasible solutions. Unlike the exact matching approaches, inexact matching is tolerant to the variations that may exist between the elements of the matched models. This makes it more practical and its result is more intuitive.

2.4 Model Merging

Model merging is the task of unifying information in the input models together while keeping a single copy of matched elements [33]. It is a kind of many-to-one transformation [62] with special requirements that are not generally required for a typical many-to-one transformation [58], and thus not supported by the general transformation tools. Within the context of our framework we state the task of our merging operator as follows. Given, as input, a set of input models along with their pair-wise correspondence, the aim of our merging task is to generate, as output, a single model, called reference model, which unifies the overlaps and explicates the differences between the elements of the input models.

2.5 The Unified Modeling Language (UML)

UML (Unified Modeling Language) [63] is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software systems. It is a de facto standard for object-oriented modeling specified by the Object Management Group

(OMG) [64, 65] . Within the context of Model Driven Development (MDD), UML, along with the Meta Object Facility (MOF) [66], provides a key foundation for OMG's Model-Driven Architecture (MDA). UML provides a variety of diagrams for modeling different aspects of software systems. For example, Use Case diagrams are used to model the system functionality, Class diagrams are used to model the system structure, and Statechart diagrams are used to model the system behavior [44].

2.6 XMI

XMI [67], which stands for Extensible Markup Language (XML) Metadata Interchange, is an interchange format for metadata that is defined in terms of the MOF standard [68]. Since UML is MOF-based meta-model, XMI can be used to represent and store UML models in XML-based interchange format [69]. This allows UML modeling tools, e.g. Altova [70], or repositories from different vendors to use XMI to exchange UML models. Thus, XMI integrates three standards: MOF (OMG), UML (OMG), and XML (W3C [71]).

2.7 Reference Models

Reference models are built to represent already existing practices or artifacts, and thus serve as blueprints for developing others [72]. They also serve as recommendations on how to solve a specific problem, means to access industry best practices, and benchmarks against which design practices are compared and evaluated [8, 72, 73]. Reference models can be developed in different forms, such as reference architecture, business process reference model, data reference model, etc [74]. The underlying motivation for reference models is the development by reuse paradigm [74].

Additionally, reference models can enlighten people about design characteristics in certain domain. Consequently, reference models enable practitioners to have a degree of confidence that their activities begins on a solid foundation [72].

2.8 Genetic Algorithm (GA)

Genetic algorithm (GA) is a population based search heuristic that mimics the process of natural selection. It has been used in the literature in different search-based problems, e.g. Quadratic Assignment Problem [75], generating a sequence of cities for the known combinatorial Traveling Salesman problem [76, 77], class diagram retrieval [78], graph matching [79], etc. At the core of the GA algorithm is the idea of maintaining a population of alternative global solutions to the search or the optimization problem in hand. The objective of the algorithm is to maximize the payoff of candidate solutions in the population against a cost function [80]. GA belongs to the larger class of evolutionary computation, which generate solutions to optimization problems using techniques inspired by natural evolution, such as selection (survival), crossover (recombination and inheritance), and mutation (diversity).

The algorithm starts with a population of randomly generated solutions, called individuals or chromosomes. Each chromosome represents a different candidate solution in a population of solutions. Each candidate solution is evaluated against a fitness function and assigned a fitness score. This fitness score is a measure of the goodness of each solution in solving the problem at hand. The fitness function is always problem dependent, for example, in a model matching problem it can be the reciprocal of the matching error (minimization), the overall sum of the similarity scores (maximization), or the number of elements passing the similarity threshold (maximization). The solutions of

the current population (also called current generation) evolve through what is called a breeding cycle, which is the heart of the genetic algorithm. The breeding cycle consists of three steps, selection, crossover, and mutation. In the selection step, solutions compete for survival in the next generation through a selection mechanism. The chance of survival is proportional to the fitness score of the solution. Typically fitter solutions are more likely to be selected. The new population is then generated from the selected portion of the current population through two genetic operators, viz. recombination (crossover) and mutation. The former operator crosses a pair of solutions (called parents) to generate new solutions (called offspring). It is supposed to exploit the current solution to find better one [80, 81]. The later operator mutates the offspring to introduce a genetic diversity between generations. It has traditionally considered as a simple search operator that helps the algorithm to avoid being trapped in the local optima [82]. It is meant to help for the exploration of the whole search space. The process of breeding new individuals from current ones and evaluating them against the fitness function is repeated until the termination condition is met.

The basic steps of GA can be sketched as the follows [81]:

1. Create an initial population of candidate solutions.
2. Compute the fitness values of each of these candidates.
3. Select candidates for new generations using some selection mechanism.
4. Make perturbation to each of these selected candidates using genetic operators, e.g. crossover and mutation.
5. Repeat 2 through 4 until the termination condition is met.

2.9 Simulated Annealing Algorithm (SA)

Simulated Annealing (SA) algorithm [83] is a local search meta-heuristic algorithm capable of escaping from local optima. The algorithm provides a probabilistic exploration for the solutions' search space. This probabilistic exploration of the solution space helps the algorithm to avoid being trapped in the local optima [84]. SA has been used in the literature in different search-based problems, e.g. graph isomorphism [85], generating a sequence of cities for the known combinatorial Traveling Salesman problem [86], grid scheduling [87], etc. It is so named because its behavior is simulating the annealing process of solids in the thermodynamic system, where a crystalline solid is heated and then allowed to cool very slowly until it achieves its most regular possible state (the ground state), which results in a solid with superior structural integrity.

The algorithm starts with initial (generally random) solution, then in each of its iteration it computes the objective function which indicates the quality of the *new* solution (also called *neighbor* solution) as compared to the *current* solution. Better solutions are always accepted while worse solution are probabilistically accepted. The acceptance probability of the worse solution is generally high at the beginning to allow for better exploration of the solution space, and thus escaping from getting trapped by the local optima with the hope to find the global optima. However this probability is decreasing over time until it reaches a point where the exploitation starts to outweigh the exploration.

[CHAPTER 3 |

LITERATURE REVIEW

3.1 Introduction

Historically, reuse in software started on the low-level technical assets, which is code. In the 1960s and 1970s, the reuse focus was at the level of subroutines [29]. The emergence of object-oriented approach, in 1980s, shifted reuse practice from libraries of isolated functions to library of classes and cross-language blocks of code [88]. In 1990s, the software reuse has been stepped to larger grain pieces, software components [88].

With the emergence of software product line (SPL) in the late 1990s, the software reuse process has been promoted from ad hoc and opportunistic to systematic [29]. Being a highly successful approach to strategic reuse, SPL have been widely adopted in the industry and the academia following two main strategies [32]: forward engineering (proactive) and reverse engineering (extractive). The proactive SPL approach emphasizes the development of the core (common) assets first. Although the engineering practices of this approach are straightforward and result in a sound product line architecture, the identification of the common assets among the different variants of the system family upfront requires a foreseeable horizon, which is seldom possible [32, 89, 90]. Therefore, the very often, yet not straightforward, practice is to extract (reverse engineer) the product line architecture from a set of legacy artifacts [89].

There are two architectural representations of the product line architecture [32]. The first approach provides a generic architecture for the product line, which captures the

commonalities of the products family but ignores all the variabilities. In this approach, each application starts with the generic architecture and adapts it as required.

The second approach, which is more desirable, explicitly captures both the commonalities and variabilities of the products family. From the reuse perspective, the first architectural representation targets reuse through specialization, as it captures the reusable knowledge and practice at a high level of abstraction. Although the abstraction level of knowledge captured by this representation promotes across domains reuse (for domains that share similar characteristics) and provides a good starting point as compared to developing a system without any reuse, it fails to capture any knowledge about the variability in a family of products [91]. Moreover, this approach requires a significant effort by experts for specialization [40]. The second architectural representation targets reuse through customization, as it aims at capturing all possible solutions and at the level of details that promotes “as-is” or direct reuse [91]. In this representation, the commonalities among the different possible solutions (artifacts) are unified and represented as common assets (core-assets) and the variabilities are explicitly modeled as alternative (mutually exclusive) or optional assets. Modeling variability in software systems has been acknowledged to be a necessity [46, 92, 93]. Variability contributes to the success of reuse in the sense that variable artifacts are modeled to capture the expected diversity in the requirements of the different products while supporting as-is reuse [92].

3.2 Model Consolidation: Opportunities and Challenges

Both Software Product Line Engineering (SPLE) and Model Driven Development (MDD) are emerging technologies that encourage software reuse. The former technology

supports reuse through providing an effective mechanism for reusing the common assets. The later technology (i.e., MDD) supports reuse through different levels of abstraction provided by the models at different stages of the development life cycle [32, 94]. Adopting the key activities of SPLE into MDD provides a systematic way to build, out of a set of existing MDD models, a reusable reference model with the following benefits [95-97]:

- It promotes the reuse practice of MDD models from ad hoc into systematic by capitalizing on the commonalities and variabilities managements of an SPLE to capture the commonalities and variabilities across MDD input models.
- MDD models will serve as a reference reusable assets, both horizontally (i.e., for similar products) and vertically (for later stage artifacts).
- Having a reference model that captures what is common and what is variable across different analysis (design) experience instances in a domain will guide the creation of new applications in that domain.
- MDD models become first-class reusable assets.
- The complexity of creating, maintaining, and evolving a set of similar artifacts will be reduced to the simplicity of a single system.
- The reference model will capitalize on the combined reuse benefits of both SPLE (such as strategic reuse, and commonalities and variabilities managements) and MDD (such as reducing cognitive distance through model's abstraction) [4].

However, building a reusable reference model out of a set of existing individual models is not a straightforward task and many issues should be taken to account [19, 37,

42, 98]. Among these issues are: assumptions about the input models; detecting the commonality and variability among different models; modeling variability on the merged model, explicating the nature of the relationship among the elements involved in the merge process; the cohesiveness of the models to be merged; resolving lexical conflicts, resolving structural conflicts; resolving semantic conflicts; resolving behavioral conflicts; providing the ability to generate the originating individual models back from the merged model, and others.

Different works in the literature have been addressing the problem of consolidating a set of existing models to build a single generic model. Bernstein et al. [99] proposes a data model on which the model management operations (matching, selection, merging, and composition) are defined. In that data model, models and mappings are first class elements. In their approach, a model is a set of objects. Every element in a model is reachable from a root object using containment relationships. Mappings are models that represent the relationships between models. Their work is an attempt with the ultimate objective of establishing a framework for general-purpose model management operators (including matching and merging). However, they highlighted a set of challenges that needs to be tackled towards achieving this objective. Some of these challenges are related to model representation, and the accuracy and the efficiency of both matching and merging operators.

Kim et al. [100] present an approach of forward engineering and re-engineering activities for building a software product line out of a set of related legacy systems in the digital audio and video domain. They interleaved re-engineering activities with the main (forward engineering) activities, where the reverse-engineering is used to extract the

candidates of core assets from the recovered architectural models while the forward engineering incrementally applies the main activities of the development process to refine these assets through analyzing the code and design documents of the legacy systems. Based on their experience they list a set of guidelines to enhance the quality of the constructed software product line and to evaluate the constructed reference architecture against these guidelines.

Breivold et al. [89] provided structured migration methods to merge legacy systems to product line architecture based on their industrial experience. In this work they list a set of recommendations for the transition process from legacy systems to the product line. This approach emphasizes the software architecture as a key to recovery of domain concept and relations.

Brunet et al. [51] proposed a framework for research on model merging, in order to be able to discuss and compare the many different approaches to model merging. They propose a set of useful model management operators (*merge*, *match*, *diff*, *split*, and *slice*) and specify the idealized algebraic properties of each operator. Using this framework, different proposals can be compared.

Lutz et al. [37] provide insights into the process of how users compare and merge visual models. The underlying question of their work is “*How do software engineers merge UML models?*” Their main contribution is the use of qualitative theory to demonstrate human model merging activities and the derived findings, as guidelines for tool design. They claim that their findings can be applied to any graph-based, visual models in software engineering. However, the focus in their work is the UML class

diagram. They also list a catalog of alternative ways to model the same or similar aspects, in an attempt to show some of the difficulties involved in the similarity assessment and the matching process which, in turn, hurt the accuracy of the merge process. The authors also highlight some factors that should be considered when assessing the similarity between models to be merged as well as a set of factors that should be considered by the merging process.

Toward standardizing model merging expectations, Barrett et al. [98] commenced the effort by assessing a set of representative merging tools. Their assessment on three merging tools (IBM Rational Software Architect, IBM Rational Rose, Sparx Enterprise Architect) to merge two versions of a simple class diagram showed that the tools “*were not up to the task*” and their performance is “*downright counterintuitive*” even for trivial models. Based on their findings they provide a set of recommendations for the tool vendors. These recommendations are meant to improve conflict detection and resolution mechanisms, and the accuracy of the merging tool.

Recently, Chechik et al. [19] differentiate three key model integration operators (merging, weaving, and composition) and describe each operator along with its applicability. Then they elaborate on the merge operator and the factors that one must consider (like, the notation of input models, formalizing the notation, assumptions) in defining a merge operator. They provide a set of criteria, such as completeness, non-redundancy, minimality, totality, and soundness, for evaluating the merge operator. Then, to show the generality and flexibility of their framework they provide a comparison between two merge operators (called, algebraic merge and state machine merge).

The focus of the aforementioned work is mainly on discussing some methodologies, lessons learned, guidelines, challenges, and requirements that should be considered by any comparison or merging algorithm, or tool.

Other work in the literature directed their effort towards proposing and developing different matching and/or merging algorithms and tools [18, 33, 41-43, 50, 55, 61, 101]. Some of these algorithms are specific to particular artifacts [41, 43] and/or specific modeling languages [43] while some others are applicable to more than one type of artifacts [39, 42] and/or more than one type of modeling language [41-43]. Additionally, these works differ in the information they consider for matching and merging the different artifacts. The following section (Section 3.3) provides a detailed comparison among these different works.

3.3 Model Consolidation Techniques

Model integration in the general sense is about building a generic model out of a set of input models [102]. Work in the literature about integration can be classified into three approaches based on the intention of the integration process [19]:

- *Merging* a set of related models to build a generic artifact [18, 33, 36, 42, 43, 50, 103, 104]. The focus of the work in this direction is to merge the input models by unifying their overlap while considering conflicts and variability among the different models. Existing approaches differ in aspects such as the merging approaches used, handling conflicts, modeling variabilities, etc. The goal is to provide better model management such as managing evolution [31, 41, 43], managing uncertainty [36, 42], avoiding redundancy, extra cost and/or targeting large-scale reuse [41, 50, 104], migration

towards product line from legacy artifacts [18, 33, 94, 100, 105, 106], and views merging [42].

- *Composing* a set of autonomous, interacting models to form one model [46] [107, 108]. Here, the input models are treated as a black-box with interfaces to the outside world and the composition is done by appropriately joining these interfaces. The goal is to deal with issues like synchronization and concurrency.
- *Weaving* a set of cross-cutting concern models into a base system model [107-109]. Here, the Aspect-Oriented concept is applied, where cross-cutting concerns are modeled as autonomous fragments and appropriately integrated into the base model. The goal here is to provide better modularity which improves the maintainability of models.

Since our focus here is the consolidation of a set of related models to build a general reference model, i.e. merging, the last two approaches will not be considered further. As mentioned earlier, creating a reference model out of a set of existing analysis (design) models involves many issues. In the following subsections we elaborate on these issues and show how they have been treated in the literature.

3.3.1 Detecting Overlaps (Commonalities) and Differences (Variations)

A fundamental operation towards efficient consolidation mechanisms is to have an efficient detection mechanism to identify the commonalities and the variabilities among the models to be merged. There are two main research streams in this area: (1) the development of *similarity measures* (matchers) that adequately capture all the necessary information about the models to be merged; and (2) the development of efficient

matching algorithms that use the similarity measures to identify identical, similar, and different elements of the models to be merged.

Similarity measures: there exist a number of similarity measures which can be classified based on the information they capture (Universal Index [43], Name [33, 36, 41, 50, 104], structure [18, 33], layout [50], semantic or role [33, 36], and behavioral [18]^{*}[50]), the level of the abstraction (schema-level [55] and instance-level [18, 33, 36, 41, 43, 50, 104]), and the level of granularity (element-level [18, 33, 36, 41-43, 50, 104] and structure-level [36, 41]).

Matching algorithms: Work in this direction can be classified into: Tree-based [110], Heuristic-based [33, 78], Clustering [41] and iterative [43]. Also some matching algorithms can be either exact match [43, 60, 111, 112] or approximate match [18, 33, 50, 61].

3.3.2 Modeling Variants

As mentioned earlier, models overlap in some elements and vary in others. Overlapped elements are unified in the generic consolidated model while variants require some mechanisms to track them, understand their differences, and to be synchronized over time. Work in this direction can fall in two classes: (1) Modeling the variants within a single consolidated model, which forms a super set capturing commonalities and variations among the set of input models [18, 33, 36, 41-43, 50, 103, 104]; and (2) Keeping the variants as separate model fragments [46] .

^{*} Just a concept, no defined measure

Modeling the variants within a single consolidated model (also known as Negative or Annotative variability): In this approach, the consolidated model is characterized by incorporating variation points to distinguish the parts that are common to all variants from those that are specific to certain variants. The idea is to minimize the effort of developing and maintaining model variants by working on a single artifact — the consolidated model — rather than on each variant separately, and then configure the consolidated model via its variation points, so as to obtain one of its input variants when needed. The key issue in this approach is how to represent the variation points. Various approaches exist in literature: (1) using configurable nodes [74, 113]; (2) marking elements with stereo-type or specific notations [18, 33, 36, 42, 50, 103]; (3) using aspect-oriented principles [108, 114, 115], (4) Using feature model notation [33, 104], (5) using abstraction [18], and (7) through ordered sequence of changes (Δ) applied to the original model [41, 43], etc.

Modeling variants as separate model fragments (also known as Positive or Compositional variability): in this approach variants are modeled as separate model fragments with mechanisms to track their commonalities [46] .

3.3.3 Merging Approaches and Algorithms

Model merging is a mechanism of combining a collection of variants into a consolidated single model. The goal of any merging algorithm is to combine the input models in such a way that their overlaps are unified to minimize the redundancy among the input models. Merging implies that a comparison of the corresponding elements has been already performed, similarities have been assessed, and rationales for their further treatment have been derived [37]. Work in this direction can be classified into two

approaches: (1) Bottom-Up-Top-Down approach [33, 36, 42, 43, 104]; (2) Bottom-Up approach [18, 41, 50].

Bottom-Up-Top-Down merging: In this approach the merge is performed by the set union of the elements in the input models (Bottom-Up). In other words all the elements in the individual models are presented in the consolidated model. Additionally, it should be possible to generate each one of the input models from the consolidated model (Top-Down). For example, in [33] a Union-merge is proposed to construct the consolidated model. Additionally, to allow the instantiation of each input model from the consolidated model, a mapping function (σ) is used to map each element in the input model M_i to its corresponding element in merged model M , and a reverse mappings (σ_1 and σ_2) are used to do the reverse (i.e. from M to M_i). In [43] merge is done through Delta, and instantiation is done through the inverse of Delta. In [42], merge is done through disjoint-set, and then refined using category-theoretic concepts like interconnection diagram and an algebraic concept called *colimit*. To provide the ability to generate the input models from the merged model, a detailed annotation mechanism (annotation-set) is used. In [41] (scenario 1), although a mechanism is presented to evolve the reference model with the aim to keep it with minimum distance from the variants, the variants need to be traced directly to the evolved reference model. Additionally, in scenario 2, the variants are clustered based on their frequency which compromises some of the variants, making instantiating the exact original instance from the reference not possible.

Bottom-Up merging: In this approach the focus is only the merge (Bottom-Up) while replaying the process downward is not considered or guaranteed. For example, in [18] the

merged model is refined to become more abstract using identity and similarity degree threshold. However, no mechanism is provided in the backward direction.

In [41], the activities are clustered based on similarity of their relations with other activities over the different variants. Furthermore, the order relation between two activities to be clustered is determined by the relation that has the highest frequency in the different variants. This results in ignoring less frequent ordered relation in the consolidated model, making tracing the corresponding variant difficult.

3.3.4 Model Assumptions

One of the issues of models' consolidation is the assumption made about the input models. Different approaches differ in the assumption they make about the model, where models are assumed to be: alternatives of the same system [36, 42, 50], multiple view with the same parent node [104], related products [18, 33], and derived from original model by a sequence of operations [41, 43].

3.3.5 Artifacts and Modeling Language Considered

Software development involves different artifacts that represent different system perspectives at different level of granularity. The artifacts that are considered by the different approaches are: Statechart only [33, 36, 50], class diagram only [43], statechart and class diagram [18], class diagram and sequence diagram [39], feature model [104], goal model and entity relationship diagram [42], process models (activity diagram) [41], etc.

As per the modeling languages for representing the software development artifacts, the matching and merging approaches can be applicable to more than one modeling

language [33, 36, 41-43, 50, 55, 112] while other approaches are specific to particular modeling language [18, 104]. In the former approaches, models are often represented as generic graphs. This representation makes the match/merge operator generic enough to be applied to different modeling languages. However, these approaches make it difficult to reason about the semantic properties of the merged model. Unlike the generic merge operators, the specific merge operators (often represented as specific graphs) provide a direct basis for reasoning about preservation of semantic properties during merge.

3.4 Matching: Technical Aspects

In this section we compare the different matching approaches in terms of the information used in the analysis and assessment of the similarities and differences, as well as the algorithms used for matching.

3.4.1 Granularity of Matching

Matching can be performed at various levels of granularity, e.g. element-level and structure-level matching [116]. In the element-level matching, a match is to be found between elements of a model and elements of another model [61]. Structure-level matching, on the other hand, refers to matching a fragment of a model (combinations of elements) with fragments of another model. A well-known example of the latter is the detection of design patterns within the design models [117-119].

3.4.2 Identification of Similarities

Similarity between models can be assessed using different strategies and similarity information. This similarity information can be classified into:

Identification of Typographic Name Similarities: This is a label-based (textual) comparison of two names to decide whether they are the same or not. However, two situations are common: 1) having two different elements with the identical names; 2) having two identical elements with textually different labels. Therefore, considering only typographic similarity to decide whether two elements are identical or not might lead to wrong matching. Consequently, this strategy should be combined with other strategies to get more accurate matching [33, 36, 104].

Identification of Lexical Name Similarities: Measures the similarity between name labels based on their linguistic correlations. This can be done through two approaches: 1) building a specific electronic synonym dictionary; 2) using one of the freely available dictionaries like WordNet [120]. This can solve the second problem faced in the context of the typographic similarity measure. However, the first problem is still present.

Identification of Layout Similarities: The main purpose of layout similarities is to identify similar elements based on their relative positions. For example, in [104], elements must be at the same level to be compared. Although, this measure is so restrictive and may result in non-optimal matching, it is desirable in some situations. Therefore, this strategy should be combined with other strategies to get more accurate matching. In [33] all nodes and edges are rooted to the same root, thus layout is not preserved. In [104], for the elements to be compared they must be in the same level and share the same parent.

Identification of Semantic Similarities: The exact definition of semantic similarities might be different from context to another, but, in general, the sense of this similarity

measure criterion is that elements are compared with respect to their roles, functionalities, or their purpose. Identification of semantic similarities is the most frequently used complex strategy.

Identification of Structural Similarities: With the structural similarity, elements are compared based on their structural properties, such as their relationships to other elements, the cardinality of fan-in and fan-out interactions with other elements, etc. However, it is often that elements may have the same structural similarity, but different functionality. Therefore, structural similarities were rarely identified explicitly; they were often used to support the other strategies, especially semantic similarities [33].

Identification of Behavioral similarity: with behavioral similarity elements are compared based on their execution semantics. Rubin and Chechik presented just the concept in this regard without proposing any metrics [18].

Identification Universal Index similarity: In this strategy, elements are compared based on a universal index. For example, in [43] elements are mapped based on universal index.

3.4.3 Handling Conflicts

Conflicts in similarity assessment are common. For example, two identical classes can be mapped to some other classes by different relationships. These conflicts need to be investigated and resolved. Their resolution can be in different ways: modeling them as alternatives (if different), merging them (if they are the same), introducing generalization (if they are parts of missing whole), favoring one over the other, etc. Guided by [37], we can list the following possible conflicts. These conflicts should be resolved in a way that

preserves their semantic and identity, meanwhile their representation in the merged model is efficient.

Handling Name conflicts: Name conflict occurs when two design elements representing the same underlying concept have different names in the input models. This can be detected through other tests that are not based on name similarities. The element in the merged model can be named randomly from the available names in the input model, or using some preference mechanism.

Handling structural conflicts: Structural conflict occurs when two design elements representing the same underlying concept have different structural properties. This can be detected through other tests that are not based on structural similarities. Resolution can be through some mechanisms like rule-based or frequency (voting).

Handling semantic conflicts: semantic conflict occurs when two design elements representing the same underlying concept have different semantics. This can be detected through other tests that are not based on semantic similarities. Resolution of this conflict can be additive (maintaining both conflicting elements) [36], or compromised (using some preference mechanism or rule-based) [104].

Handling layout conflicts: Layout conflict occurs when two design elements representing the same underlying concept have different depth. This can be detected through other tests that are not based on layout similarities. One way to handle such a conflict is through generalization.

Handling behavioral conflicts: Behavioral conflict occurs when two design elements representing the same underlying concept have different execution behavior. This can be

detected through other tests that are not based on behavioral similarities. One way to handle such a conflict is through frequency, another way is through annotated branching [18].

Handling missing (unmatched) conflicts: Missing conflicts occur when an element exists in only one input model, but not in the others. Missing elements can be handled in many ways. One way is to just add the elements that do not match in other models to the consolidated model [36]. Another way for those elements is to be compromised through a rule-based [104] or similarity-based [18] mechanism. A trade-off may result in a different semantic representation in the merged model.

Handling Design conflicts: Design conflicts occur when the same concept is modeled differently. For example, a system feature can be modeled differently in two different models. Another example, the same feature can be modeled at different level of details in the two different models. Third, a system feature can be modeled by one design element in one model while distributed over different design elements in other model. One way to handle such conflict is through three-way merging. Another way is through a well-known optimal solution (e.g. design patterns) for such system feature.

Considering multiple system views: Software development involves a set of diagrams to model different aspects of systems; for example, functional, structural, and behavioral aspects. These diagrams should be consistent with each other and the information available in one diagram should help in discovering the missing information in the other diagram. Having a matching (merging) algorithm that considers the information available in these different views of the system will make the result of match (merge) more

accurate. In this regard, work in the literature can be classified into those matching (merging) mechanisms who consider only one view [50, 60, 61, 101, 111, 112], two views (structural and behavioral) [18], and three views.

Similarity levels: Similarity levels refer to the number of levels to which the similarity scoring is graded. It is the number of thresholds between grades plus one. For example, in [18] there are two levels (identical, similar) while in [41] there is only one similarity level (similar).

To recap, Table 1 summarizes and compares different approaches proposed in the literature in terms of the similarity information used for matching software artifacts. Our proposed metrics are indicated in the last row of the table.

Table 1. Matching Approaches: Similarity Information Used

	Typographic similarity	Lexical similarity	Structural similarity	Semantic (Role) similarity	Layout Similarity	Behavioral similarity	Universal Index	Views considered	Matching granularity	Matching Abstraction	Tool support	Artifact
Fam12[36]	✓*	×	×	×	×	×	×	1	E/S	I	✓	State Machine
Ayd11[104]	✓*	×	×	✓	✓	×	×	1	E	I	×	Feature model
Ala03[43]	✓	✓	✓	✓	×	×	✓*	1	E	I	×	Class diagram
Nej07[50]	✓	✓	×	×	✓	✓	×	1	E	I	✓	State Machine
Li11[41]	✓*	×	×	×	×	×	×	1	E/S	I	✓	Activity
Rub10[18]**	×	×	✓	!	×	✓	×	2	E	I	×	Class + State Machine
Rei10[39]	✓	✓	✓	✓	×	✓	×	3	E	I	✓	Class + Sequence
Our	×	✓	✓	✓	×	×	×	1	E	I		Class
	E S I	Element Structure Instance of model			✓ × !	Property is supported Property is not supported Implied		* ? **	Main measure Under some conditions Just conceptual			

3.5 Merging : Technical Aspects

In this section we compare the different merging approaches based on the following criteria:

Dealing with weak variants: Weak variants can be at fine-grained level (e.g., model elements) or coarse-grained level (e.g., model variants). A variant is weak in the sense that it has low preference value or weight under some evaluation mechanism. For example, in [41] variants are weighted based on the number of instances that were created from each one of them. Dealing with such variants can be through adding them to the consolidated model [36, 43] or they can be compromised through some mechanism [41, 104]

Noncritical Differences: Noncritical difference between the elements of the input models can be either modeled as variants [36, 42, 50], or compromised by rule-based mechanism [104], or based on threshold [18]. In the later approach weak elements might not be represented in the consolidated model. This may make the instantiation of some variants from the consolidated model difficult or even impossible.

Way of merge: Merge can be either two-way merge or three-way merge. A two-way merge compares two models and merges them into a single model. A three-way merge, on the other hand, requires access to a baseline model (or fragment) that serves as a reference to both models.

Completeness: If an element appears in one of the source models, it must be represented in the merged model as well. This is to ensure that information in the source models is preserved in the consolidated model. This property is assured by some merge

algorithms [36, 42, 43, 50], but not assured by others [18, 104]. In [104], this property is not assured, because when resolving the conflicts through some rules some variants might be compromised, thus not represented in the merged model. In [18], reducing the complexity of the merged model, through a reduction of similarity and identity threshold, results in missing information about the source variants.

Non-redundancy: If an element appears in more than one source model, only one copy of it is included in the merged model. For example, this property is assured in [18, 36, 42], but not in [43, 50, 104]. For example, in [43], redundancy comes from the fact that identical elements with different IDs cannot be detected as identical, and thus more than one copy of the same element can appear in the merged model. In [50], the definition of the shared transition similarity is conservative. This may result in redundant transitions.

Minimality: Merge must not introduce new information, which is neither present nor implied by the source models. This property may be in contradiction with the conflict resolution mechanism, where information may be added or deleted to resolve conflicts.

Totality: Merge can be performed for an arbitrary set of models. This property is of particular importance if one wants to tolerate inconsistency between the source models. For example, this property is assured in [36, 42, 50], but not in [41, 43]. For example, in [43], this is not assured due to the fact that source models are assumed to be derived from an original model by a sequence of changes.

Soundness: Merge must preserve the semantic properties of the models to be merged. This property is assured in [36, 43, 50], but not in [18, 42, 104]. In [42, 104], this

property is not usually preserved as it depends on the rule of merging different level of knowledge. In [18], the merge process is iterative, and in each iteration the merged model gets more abstracted, resulting in losing more semantic information. In [50], the merge is behavior-preserving. However, preserving the static semantic is not guaranteed.

To recap, Table 2 summarizes and compares different merging approaches proposed in the literature.

Table 2. Merging Approaches

	Completeness	Totality	Targeting Reference	Dealing with weak variants	Uncritical Differences	Instantiate-ability	Proof of concept tool
Fam12[36]	✓	✓	×	A	A	×	✓
Ayd11[104]	×	×	×	RB	RB	×	×
Ala03[43]	✓	×	×	A	A	✓	×
Sab06[42]	✓	✓	×	×	A	✓	✓
Nej07[50]	✓	✓	×	×	A	✓	✓
Rub10[18]	×	✓	✓	×	M	×	×
Li10[41]	×	×	✓	RB	M	×	✓
Rei10[39]	×	✓	✓	RB	M	×	✓
Our	✓	✓	✓	A	A	✓	✓
Keys:	✓	Supported			A	Additive	
	×	Not Supported			RB	Rule-Based	
	*	Behavioral preserving			M	Metric threshold	

3.6 Summary

As we have seen in this chapter, the task of building a reusable reference model out of a set of input models is not a trivial task and many issues are involved. Models to be consolidated needs to be cohesive enough, commonality and differences between their elements must be accurately identified, conflicts must be resolved, and commonality and differences must be explicated in a way that encourages as-is reuse. Different researches tried to approach the problem, with different intentions, considering different types of

information, and using a variety of algorithms. However, there have not been enough attention paid to address the problem of automatically consolidating a given a set of analysis (design) models representing different applications (instances) in a domain into a reference model that represents the input models and promotes as-is reuse. Most of the works are entirely conceptual [18, 19, 37]. Others are directing the consolidation process towards some specific goals like resolving conflicts [36], versioning [43], rather than building a reusable reference model. Even those works who had the intention of the reference [39, 41] they focus on building the core assets in the reference while compromising the variants during merging. This will require an instantiation effort and the involvement of the experts during the instantiation [92].

CHAPTER 4

PROPOSED FRAMEWORK

4.1 Introduction

This chapter gives a conceptual description of the proposed solution framework for generalizing a set of input models into a reference model. Throughout the chapter we will be focusing on the big picture of the framework rather than the finer-grained details, which will be thoroughly presented in the following chapters. Section 4.2 lists our research questions and objectives. To give a better understanding about the interaction of its component, the framework is described through an illustrative example, which is introduced in Section 4.3. The framework components are explained with the help of the illustrative example in Section 4.4. The chapter is summarized in Section 4.5.

4.2 Research Questions and Objectives

The objective of this thesis is to develop a reference model that captures the commonalities and variabilities across the different class analysis and design instances in a domain, so that the consolidated model offers the reuse potential across the different models while maintaining the complexity at the level of a single model.

Towards this objective, this thesis, specifically, addresses the following research questions:

1. How can UML structural models be consolidated into a reference model that represents them best?
2. What metrics are needed to identify commonalities and variabilities across different input models?

3. What algorithms can be used for matching the elements across the input models?
4. How can the generalization algorithms handle the different similarity levels across the input instances and at different level of granularity?
5. How can the commonality and the variability between the elements of the input models be represented in the reference model at different level of granularity?
6. Does the reference model improve the opportunity of reuse?

Table 3. Handling the Research Questions throughout the Thesis' Chapters

	Chapter 1	Chapter 2	Chapter 3	Chapter 4	Chapter 5	Chapter 6	Chapter 7	Chapter8			
								Section 8.5	Section 8.6	Section 8.7	Section 8.8
Research Question 1				✓							
Research Question 2				✓	✓			✓	✓		
Research Question 3				✓		✓			✓	✓	
Research Question 4				✓			✓				✓
Research Question 5				✓			✓				✓
Research Question 6							✓				✓

4.3 Illustrative Example Description

Figure 1 shows the class diagrams of four instances of a simplified flight booking systems adopted from [44]. The models have been kept deliberately simple for clarity, but we believe that they are sufficient enough to convey how the different components of our proposed framework are applied to these instances to generate the reference model.

Being different instances within the same domain, they share commonalities and maintain some differences among them. For example, inspecting the four instances we find that all the models have a class called either “Airlines” or “Airways” representing the same underlying concept or real world object (Airlines’ company).

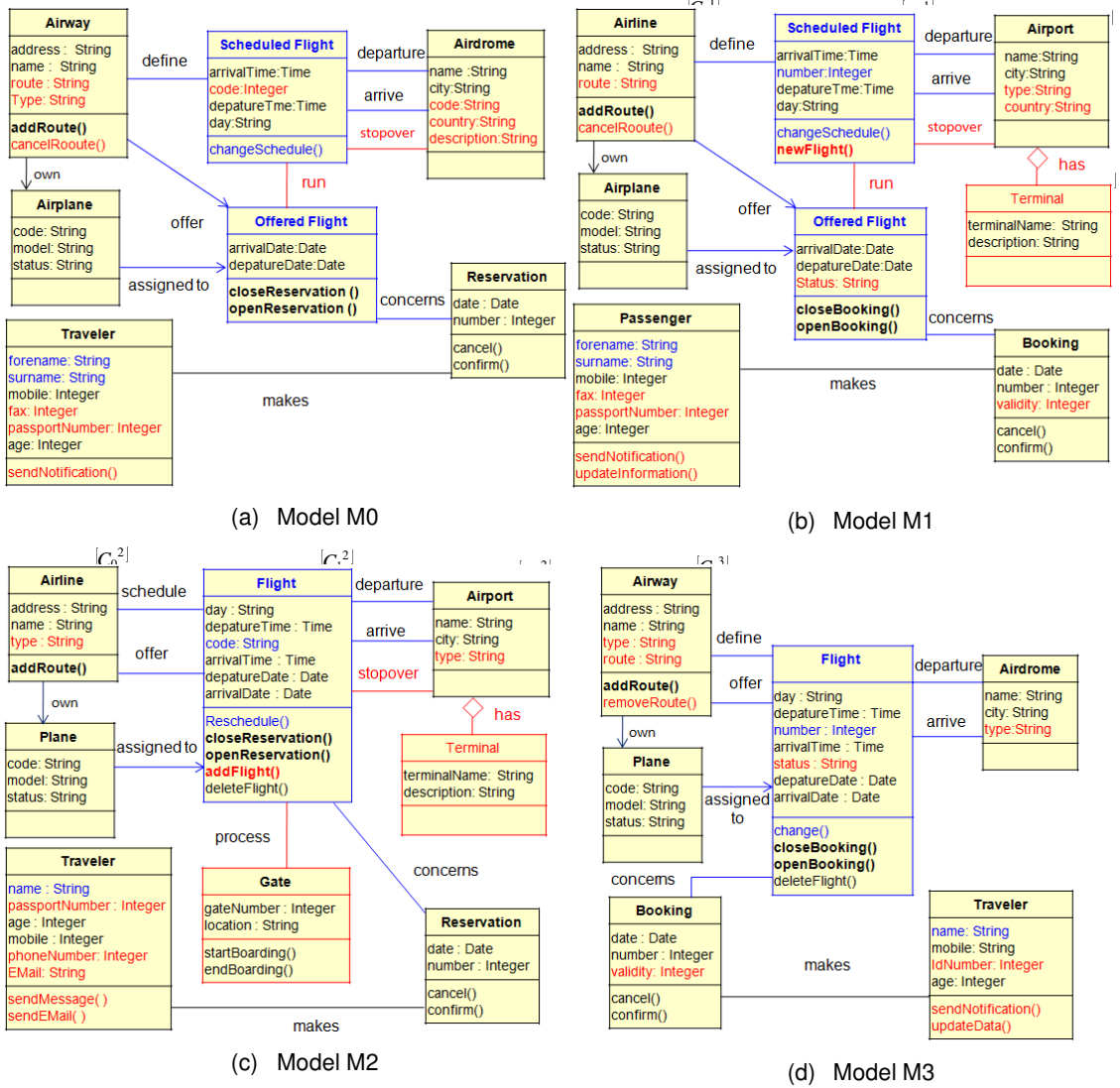


Figure 1. Input Models Representing Four Different Applications of Flight System

The textual difference between the words “Airlines” and “Airways” should not make the two words as dissimilar, because they carry the same meaning and refer to the same underlying concept. However, looking deeper to “Airlines” or “Airways” class over the four instances we can see some differences here and there. For example, considering the class attributes and their data types, the attributes “name” and “address” are common in all the instances. This is not the case with other attributes, e.g. “type” and “route”, which show up in some instances but not in the others. Some other slight differences can also be

observed considering the class methods and their parameters^{*}, and also when considering the class neighborhood.

Commonality and differences between the instances can be at different level of granularity (classes, relationships, attributes, methods, data types). Differences can be classified as either variants or optionals. Variants represent the design differences for the same underlying concept. It is present, with some design differences, in all the instances. For example the class “Flight” of M_2 (also of M_3), modeled in M_0 (also in M_1) as two classes “Scheduled Flight” and “Offered Flight”, reflecting the fact that they are two variants representing the same underlying concept. However, the class “Terminal” exists only in M_1 and M_2 but not in the other instances. Therefore, it is considered as optional.

The classes in the different models are labeled with the notation C_x^y , where the subscript x indicates the class index while the superscript y indicates the model index. This notation will be used throughout the sequel to refer to the corresponding class.

Figure 2 depicts the reference model which is the targeted output of our framework. In the reference model, common elements are unified, variants are represented as alternatives under variation points (e.g. $VP0$), and optional are represented as different options under the optional points (e.g. $OP0$).

^{*} Methods parameters are not shown in the diagrams for the sake of making the diagrams simple to show the big picture.

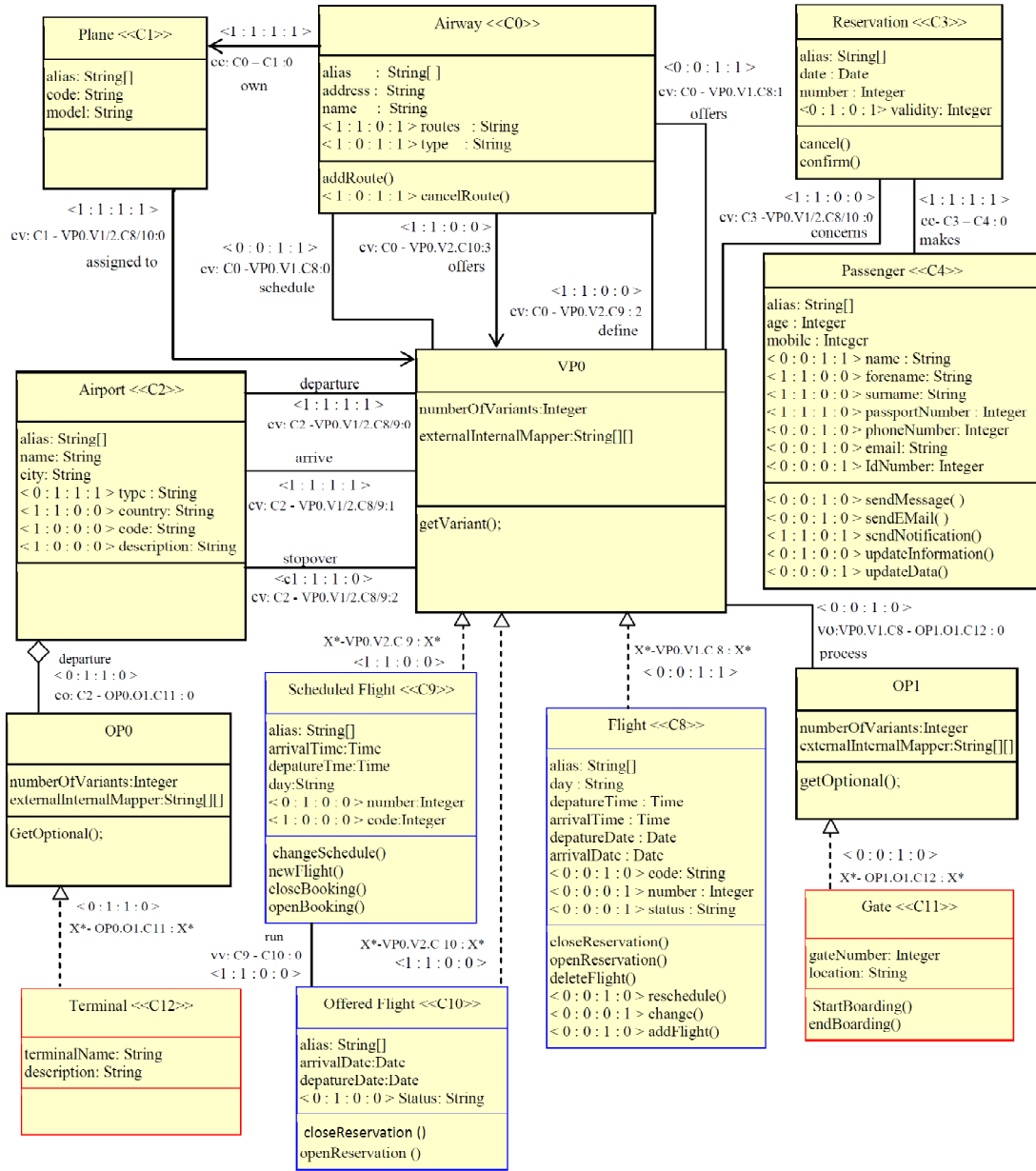


Figure 2. Reference Model for the Simple Flight Booking System

4.4 Solution Framework

Figure 3 depicts an exemplified framework of the proposed solution. Developing a reference model out of a set of input models consists of the following sequence of activities:

- (1) Parsing the input models to extract their information.
- (2) Assessing the degree of similarity between the input models, in pair-wise manner.
- (3) Matching the most related elements of the input models, in pair-wise manner, so that identical, similar, and dissimilar elements are identified.
- (4) Filtering out unrelated models, so that the reference model is cohesive enough.
- (5) Generalizing the input models by unifying their overlaps and explicating their differences in a single reference model.

The aforementioned activities can be renamed, respectively, as parsing, comparison, matching, filtering, and merging. Parsing and filtering are considered as preprocessing activities performed at different phases in the framework and they are pre-requisites for the activities following them. Comparison, matching, and merging are the three main activities in the framework, where comparison is a pre-requisite for matching, and matching is a pre-requisite for merging.

4.4.1 Parsing the Input Models

In this preprocessing task the input models, given as XMI files, are parsed to extract their information. We developed a Java-based parser that takes the XMI file(s), as input, and produces, as output, the model information to be used as input to the similarity assessment algorithms, i.e. the comparison algorithm. The parser supports two visual modeling tools, Altova and ArgoUML.

4.4.3 Staged Model Matching

During the matching, elements of each pair of models are mapped in pair-wise, based on their similarity scores, so that elements representing the same underlying concepts should be matched together. Due to the possible design difference that may exist between any two models M_i and M_j , it is possible that an element from M_i , representing certain domain concept, can be matched to either one or more elements from M_j , representing the same domain concept. For Example, referring to Figure 1, the class “Flight” of M_2 is modeled as two classes (“Scheduled Flight” and “Offered Flight”) in M_0 . Since the one-to-one matching will match the class “Flight” to only a single class from M_0 , this design difference will not be fully captured, i.e. the class “Flight” will only be matched to either the “Scheduled Flight” class or the “Offered Flight” class. Therefore we propose a 3-stage matching mechanism.

Given the *ES* matrix as produced by the model comparison algorithm, the first stage matching algorithm (detailed in 6.2) finds the best match between the elements (classes) of the corresponding pair based on their similarity scores in the *ES* matrix. In this stage of matching, each class in the smaller model (the one with less number of classes) is matched exactly to one class in the other model of the pair. Genetic Algorithm (GA), Simulated Annealing, and greedy heuristics, to be detailed later, are used to make this match optimal. The optimality in this context means that each class in one model is matched to its most similar class in the other model. The output of the matching algorithm is the Matching Similarity Matrix, referred to in Figure 3 as **MSM** Matrix.

Those classes, not passing an arbitrary one-to-one similarity threshold, go through the second-stage matching algorithm, detailed in Section 6.3, where a single class from

certain model (say M_i) can be matched to many classes in another model (say M_j). Although in this stage a single class of one model of the pair can be matched to multiple classes in another model, it does not capture the situation where multiple classes in one model (representing certain domain concept) are matched to multiple classes in another model (representing the same concept). Therefore a third stage similarity assessment is proposed to handle such situations, where the residuals (classes not mapped yet) are to be added to the most appropriate class group based on their contribution to the improvement of the similarity scores. Detailed description about third stage similarity assessment algorithm is given in Section 6.4. Since the matching in the second and the third stages involves a group of classes matched to a single class or another group of classes, we refer to such matching as a class-group matching.

Doing the matching in a staged way has threefold objective. First, it distributes the search space of matching over the three stages. This will reduce the matching complexity. Dealing with models representing instances within the same domain is expected to have high commonality, and thus the matching of the majority of the elements will be done within the first matching stage in a polynomial time, which is also followed by another polynomial time matching stage. Therefore, only few residuals will be investigated in the third stage, which is more complex, yet still polynomial. This is actually the gain of the staged matching algorithm, i.e. reducing the time complexity through stage matching.

Second, the staged matching gives the ability to use the appropriate similarity metrics and matching algorithms in accordance with the objective of each stage. For example, in the first stage the focus is to find class-to-class match, i.e. in each pair of models (M_i, M_j) each class in the smaller model (say M_i) will be matched to exactly one class in the

larger model (say M_j). This also means that the similarity between the classes to be matched should be based on the information that characterizes a single class (e.g. class name, class attributes and their data types, class methods and their signatures) rather than a group of classes, which is the case in the second stage. Therefore, the neighborhood information, despite its importance, may not add much to the similarity between the two matched elements, especially if we consider its cost. This last statement, as will be demonstrated by our empirical investigation, is especially true when the matched models are within the same application domain. On the other hand, in the second stage, the importance of the neighborhood information is emphasized, where the elements are matched based on their internal characteristics (i.e. attributes along with their data types, and methods along with their signatures) as well as their surroundings (i.e. neighbor name, relation name, and relation type).

Third, since our ultimate goal is the consolidation of the input models to a single reference model which unifies their commonalities and explicates their differences, staged matching allows us to perform matching with an eye on merging activities, where each matching activity can be aligned with an activity in the merging phase. For example, in the first matching stage classes are matched on one-to-one basis. Highly similar classes means that the two classes are almost identical. This means that they can be represented as a single class in the reference model. The job of the merging algorithm then is to watch for this commonality across all the pairs, and also to deal the lower granularity of commonality and differences.

To explain the concept of the staged match with example, let Table 4 represent the *ES* matrix between the classes of M_0 and M_2 . Note that in the table we present the class

notation (C_x^y) as well as the class name. In the discussion sometimes we will use the class notation rather than the class name unless it is necessary to mention the class name for clarity.

Table 4. *ES* Matrix, Pair-wise Similarity Between M_0 's Classes and M_2 's Classes

			Model M_2							
Model M_0			Airline	Flight	Airport	Reservation	Traveler	Terminal	Plane	Gate
			c_0^2	c_1^2	c_2^2	c_3^2	c_4^2	c_5^2	c_6^2	c_7^2
	Airway	C_0^0	0.83	0.35	0.4	0.48	0.38	0.22	0.39	0.43
	Scheduled Flight	C_1^0	0.45	0.52	0.37	0.34	0.23	0.16	0.23	0.29
	Airdrome	C_2^0	0.46	0.29	0.84	0.3	0.24	0.45	0.54	0.25
	Reservation	C_3^0	0.38	0.3	0.27	0.97	0.4	0.24	0.31	0.46
	Traveler	C_4^0	0.37	0.22	0.22	0.33	0.85	0.26	0.25	0.45
	Offered Flight	C_5^0	0.37	0.5	0.35	0.43	0.3	0.17	0.28	0.37
	Airplane	C_6^0	0.32	0.2	0.49	0.27	0.19	0.49	0.97	0.23

When *ES* matrix has been given to the matching algorithm, it matched the classes as depicted in *MSM* Matrix (Table 5). The similarity values of the matched classes are also shown in *MSM* Matrix. As shown in the *MSM* matrix, some pairs are matched with high similarity scores while others are matched with low similarity scores. For example, the “Flight” class (C_1^2) of M_2 is matched to the “Scheduled Flight” class (C_1^0) of M_0 with a similarity score of 0.52. This is relatively a low similarity score. We can also notice that the “Gate” class (C_7^2) of M_2 is matched to the “Offered Flight” class (C_5^0) of M_0 with similarity score of 0.37. The former match (i.e. C_1^0 to C_1^2) is partial while the later (i.e. C_5^0 to C_7^2) is totally wrong match. This fact is reflected in the corresponding similarity scores of the two matches. These two low scored matched pairs will be filtered out by the first stage similarity threshold, as not appropriately matched, and they are passed to the second matching stage.

Table 5. MSM Matrix, the Matched Classes' Similarity

M_0 classes	C_3^0	C_6^0	C_4^0	C_2^0	C_0^0	C_1^0	C_5^0
M_2 classes	C_3^2	C_6^2	C_4^2	C_2^2	C_0^2	C_1^2	C_7^2
Sim. Score	0.97	0.97	0.84	0.83	0.85	0.52	0.37

In the second stage, those elements not passing the one-to-one similarity threshold, in the first stage, need to be further investigated for potential similarity through more complex matching process. In this stage a single class from certain model can be matched to more than one class in another model. In particular, referring to our example, let S_0 and S_2 be two subsets of classes in M_0 and M_2 not passing the first stage similarity threshold, which is assumed to be usually high so that matched classes passing such a threshold will be considered as highly similar (or, metaphorically, identical). Assuming a threshold of 0.80, then:

$$S_0 = \{ C_1^0, C_5^0 \}$$

$$\text{and } S_2 = \{ C_1^2, C_5^2, C_7^2 \}.$$

As we can see here, the majority of the classes from the two matched models have passed the first matching stage threshold. This means that majority of the matching has been identified by the first stage matching algorithm and only few classes will be considered by the second-stage matching algorithm.

When S_0 and S_2 are given to the second stage matching algorithm, it will re-evaluate their similarity based on their neighborhood information and based on their internal structure. Table 6 shows an exemplifier of the second stage similarity matrix, called Group Similarity matrix (GS), between the classes of S_0 and the classes of S_2 . Some possible grouping is done by combining the information of more than one class into a single similarity *Class-Group*. For example, referring to our example, it is clear from

Figure 1 that the two classes “Scheduled Flight” and “Offered Flight” of M_0 , combined, have exactly the same neighborhood and similar internal structure (attributes and operations) as the class “Flight” of M_2 . Therefore, when comparing the similarity of the two classes, “Scheduled Flight” and “Offered Flight”, combined, against the class “Flight”, we got a similarity score of 0.76. This is a high similarity as compared to the similarities obtained when comparing each class alone against the class “Flight”, which are 0.45 and 0.42 for the classes “Scheduled Flight” and “Offered Flight”, respectively. It is worth mentioning here that for two classes to be combined, they must be adjacent to each other. For example, the combination of the two classes “Flight” (C_1^2) and “Terminal” (C_5^2) is not applicable (N/A).

Table 6. Second Stage Similarity Matrix (GS Matrix)

		M_0 classes/class-groups		
		C_1^0	C_5^0	$\{C_1^0, C_5^0\}$
M_2 classes/class-groups	C_1^2	0.45	0.42	0.76
	C_5^2	0.10	0.11	0.09
	C_7^2	0.31	0.46	0.27
	$\{C_1^2, C_7^2\}$	0.45	0.38	N/A
	$\{C_1^2, C_5^2\}$	N/A	N/A	N/A
	$\{C_5^2, C_7^2\}$	N/A	N/A	N/A

The *GS* matrix is given to the second stage matching algorithm, which is a greedy-based algorithm whose steps are depicted and exemplified in Figure 4. The intuitive assumption underlying this algorithm is that a pair of classes/class-groups with the highest similarity values is the most relevant pair. Given the *GS* matrix between the classes / class-groups of two models, the algorithm looks for the highest similarity score, in the *GS* matrix, for which the corresponding pair of classes/class-groups are not matched so far. Then the algorithm matches the corresponding classes/class-group and

marks them as matched, conditioning that their similarity score passes the second stage matching threshold.

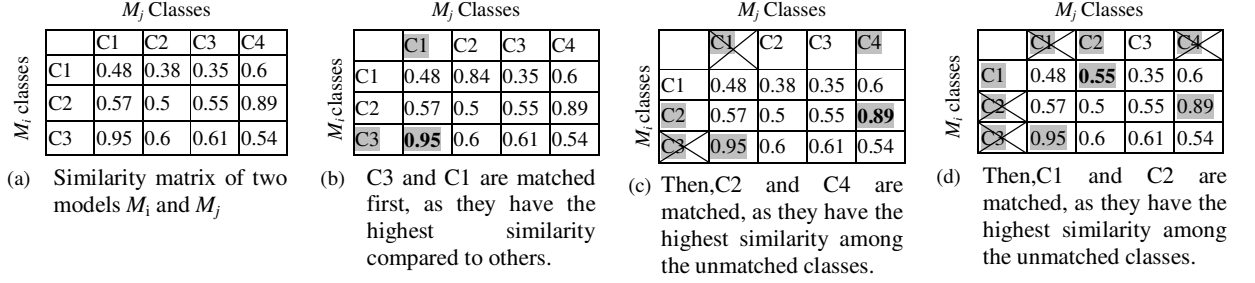


Figure 4. Steps of the Second Stage Greedy Matching Algorithm

Referring to Table 6, and assuming a second stage matching threshold of 0.75, the algorithm will match the class C_1^2 with the class-group $\{C_1^0, C_5^0\}$ as they have the highest similarity value, which also passes the threshold. Since there is no more feasible match (as there are no more unmatched classes in M_0) the algorithm will terminate.

Informal steps of the second stage matching algorithm can be sketched as follows.

1. Evaluate the similarities of the unmatched elements.
2. Do the possible grouping and evaluate the class-groups similarities and store them in GS Matrix.
3. Apply the greedy algorithm to select the highest $GS_{i,j}$
4. If $SG_{i,j}$ satisfies the threshold then match the corresponding classes/class-groups and mark them as matched and go to step 5; otherwise terminate
5. Repeat steps 1 to 4 until no more feasible match.

The formal description about the second stage comparison and matching algorithm will be presented in the next chapters. It is worth mentioning here that those classes passing the first similarity threshold are marked in the MSM matrix as highly similar,

denoted as “S”, while classes passing the second (and also the third) stage similarity threshold will be marked as variants, denoted as “V”.

Therefore, the pair of class/class-group (C_1^2 , $\{C_1^0, C_5^0\}$) will be marked as variants, as shown in the extended version of the *MSM* matrix (Table 7).

Table 7. MSM Matrix, the Matched Classes Similarity After 2nd Stage Matching

M_0 classes	C_3^0	C_6^0	C_4^0	C_2^0	C_0^0	$C_{\{1,5\}}^0$	-	-
M_2 classes	C_3^2	C_6^2	C_4^2	C_2^2	C_0^2	C_1^2	C_5^2	C_7^2
Sim. Score	0.97	0.97	0.84	0.83	0.85	0.76	-	-
Sim. level	S	S	S	S	S	V	?	?

The third stage is an extension of the second stage, where each residual class from the two matched models is considered for combining it with a class group for which the similarity with the corresponding, already matched, group is improved. In our example, C_5^2 and C_7^2 are the only residuals. Since C_5^2 is not a neighbor of C_1^2 it will not be considered for combining with it. However, when combining C_7^2 with C_1^2 similarity between the resulting class-group ($\{C_1^2, C_7^2\}$) and the class group $\{C_1^0, C_5^0\}$ is evaluated to 0.73 which is less than the similarity between the class-groups $\{C_1^2\}$ and $\{C_1^0, C_5^0\}$. Therefore the class C_7^2 as well as the class C_5^2 are considered as unmatched, marked with “U” in the *MSM* Matrix (Table 8).

Table 8. MSM Matrix, the Matched Classes Similarity After 3rd Stage Matching

M_0 classes	C_3^0	C_6^0	C_4^0	C_2^0	C_0^0	$C_{\{1,5\}}^0$	-	-
M_2 classes	C_3^2	C_6^2	C_4^2	C_2^2	C_0^2	C_1^2	C_5^2	C_7^2
Sim. Score	0.97	0.97	0.84	0.83	0.85	0.76	-	-
Sim. level	S	S	S	S	S	V	U	U

4.4.4 MSM Matrix

The *MSM* matrix is the actual output of the 3-staged similarity assessment and matching algorithms. It acts as an interface between the matching algorithms and the

merging algorithms. As we have seen, during the comparison and matching stages, the models are matched pair-wise, and based on the similarity thresholds in each of the three matching stages three levels of similarity among the matched elements of each pair of models are considered. These similarity levels, are *highly Similar* (“S”), *similar with Variation* (“V”), and *Unmatched* (“U”). The last row of *MSM* in Table 8 is depicting the three similarity levels.

The commonalities and the variabilities between the models of each pair are identified based on these levels of similarity, where elements with similarity level “S” are considered as *common*, elements with similarity level “V” are considered as *variants*, and elements with similarity level “U” are considered as *optionals*.

Having n input instances, the pair-wise matching among these instances will result in $\frac{n(n-1)}{2}$ *MSM* matrices, one for each pair of models. Referring to our illustrative example, the matching will produce $\frac{4(4-1)}{2} = 6$ *MSM* matrices to be used by the merging algorithms.

4.4.5 Filtering out Unrelated Models

As shown in the framework (Figure 3), the next activity after the pair-wise matching is to filter out unrelated models. The purpose of this preprocessing activity before starting merging is to filter out those models that can render merging infeasible. Having unrelated models consolidated to a generic reference model, hurts, in addition to other quality aspects, the cohesiveness of the consolidated model. Additionally, according to [121], “...it is worth considering the development of a family of systems when there is more to be gained by analyzing the systems collectively rather than separately—that is, when the systems have more features in common than features that distinguish them.”

Following such thought, models whose average similarity to the other models is less than 70% are excluded from being merged. We devised an algorithm to filter the unrelated models, one model at a time; re-evaluating the average similarity of each model to the others and filtering out the one with the lowest average not passing the threshold. The filtering process is repeated until the average similarity of each of the remaining models with other models is above the threshold.

4.4.6 Models' Merging

The information collected and presented in each of the $\frac{n(n-1)}{2}$ *MSM* matrices about the matched elements of each pair of models should make building the reference model a very smooth and straightforward procedure. The basic underlying process for our proposed merging algorithm can be described, as follows. Common elements in the reference model are those elements mutually have “S” similarity level across all the pairs and they are represented by a single class in the reference model. Variants are modeled through Variation Points (VP) which act as interfaces for their different variants. Optional elements are modeled through Optional Points (OP) which act as interfaces for the different optional elements. Each input model has a variant in each variation point, but it is not necessarily for each optional point to have an optional element for each input model.

Merging is performed in two phases. Each phase is implemented in a staged manner. The focus of the first stage is to perform preliminary merging at the class level, producing a reference model preliminary catalog (RMPC) in which all the common, variant, and optional classes are identified across all the instances. The RMPC acts as a foundation for

the second phase in which the union merge is performed at the level of attributes, methods, and relationships. The output of the second stage is the reference model catalog (RMC), from which the reference model, exemplified in Figure 2, is produced. Detailed description about the merging algorithms in both phases will be the focus of Chapter 7.

4.5 Summary

In this chapter we conceptually, with an illustrative example, stepped through the different components of our proposed solution for building a reference model out of a set of instances. The focus of the chapter was to draw the whole picture of the proposed solution framework and provide the reader with the conceptual roadmap before diving into the technical details which are the focus of the next chapters.

CHAPTER 5

SIMILARITY ASSESSMENT

5.1 Introduction

A fundamental operation towards efficient consolidation mechanisms is to have an efficient identification mechanism to identify commonalities and differences among the different instances to be merged. This identification task is time consuming and error-prone, especially when we have a large number of instances and/or of large size. It is error-prone due to the fact that these models, while representing similar functionalities, are modeled independently by different developers, and thus inconsistency, design differences, and intra-conflicts are expected. Therefore, their similarity and differences must be accurately quantified to have an accurate identification. The task is time consuming due to the fact that finding the similarity of two models is commonly referred to as model matching which is a kind of graph matching problem known as combinatorial problem [122]. Therefore, an efficient comparison algorithm is required to obviate this complexity of the brute-force method and meanwhile provide near (if not) optimal solution. In this chapter we will cover the first facet of the problem, i.e. the issues related to the similarity metrics. The second facet of the problem, i.e. the complexity of the matching algorithms, will be covered in the next chapter. This chapter is organized as follows. In Section 5.2 we discuss the different similarity aspects related to model comparison. Technical definitions for the similarity metrics are presented in Section 5.3. Section 5.4 lists the class level metrics. Tool support for the model comparison is outlined in Section 5.5 followed by the chapter summary in Section 5.6.

5.2 Similarity Aspects

As mentioned in the previous chapter, comparison is a pre-requisite for matching, and matching is a pre-requisite for merging [123]. Model comparison is the task of assessing or quantifying the degree of the similarity between the elements of the compared models [52, 53]. Crucial to an efficient similarity assessment is to have a set of similarity metrics that considers the various aspects of the compared models, thus their overlaps and differences are best quantified. Toward this aim, we use three types of similarity information: shallow lexical information (also called shallow semantic [124] or coarse-grained [37]), internal information (also called deep semantic [124] or fine-grained [37]), and neighborhood information. The shallow lexical information is used to measure the lexical naming similarity between the compared elements (classes). The internal information is used to measure the element's properties (i.e., attributes) and behavior (i.e., operations) similarity. The neighborhood information is used to measure the similarity of the compared elements based on their structural relationships with their neighbors.

Using either of this information individually to capture the similarity between the elements of the compared models may not usually lead to an accurate assessment. For example, two classes may have similar names, but they may totally have different properties and behavior, and vice-versa [37]. Therefore, relying on the naming similarity may not be enough to decide whether two classes are similar or not. Additionally, when the models to be compared are within the same domain, we expect the lexical similarity score between the names of the compared elements to reflect, to some extent, their real

similarity. However, this might not be the case when the compared models are across domains, as each domain has its own ontology.

Similar argument can be said when relying only on the internal information. The confounding effect of generic attributes (e.g. name and ID) and generic methods (e.g. setters and getters) can misleadingly affect the accuracy of the metrics in capturing the actual similarity between the elements of the compared models. This ultimately will lead to a wrong match. This can also happen when relying only on the neighborhood information, as two dissimilar classes from two different models may have similar, or even identical, neighbors, and vice-versa.

Using a combination of these similarity information provides complementary insights about the compared elements and allows us to consider different similarity aspects at the same time, and thus it is expected to result in a more accurate assessment. However, one of the main issues of the compound metrics is the weights assigned to each constituent of the metric [125].

5.3 Similarity Metrics

The similarity between models is quantified using a set of similarity metrics. The values of these metrics are computed based on the information collected from the compared models. In all of the metrics, concepts (classes' names, operations' names, attributes' names, and names of the relations between classes) are compared based on their semantic similarity (e.g. synonyms, hyponyms^{*}) according to the WordNet [120] *is-a* hierarchy of concepts. Relation types (as part of neighbor information) are compared using the similarity information presented in Table 10, which is inspired from [45].

^{*} e.g. tree (more specific) is a hyponym of plant (more general).

An alternative way for comparing two strings would be to use their edit distance, the minimal cost of operations to be applied to one of the string in order to obtain the other one. However, this approach is suitable for measuring similarity between strings that may contain typos, acronyms, spelling mistakes, etc [126]. It does not help when comparing two synonyms representing the same concept with different textual strings.

There are a number of measures proposed in the literature to measure the semantic similarity between two concepts. Some of these measures are based on the notion of information content [127] while others are based on the path length [128]. Content-based measures are concerned with how specific a concept is in a given ontology while path-length measures rely on the distance between two concepts counted as the number of edges (or nodes) on the path linking the two concepts [129]. The former is influenced by the corpus used. However, the later measures are independent of corpus statistics, and thus uninfluenced by sparse data [130]. Path length between two concepts can be measured in different ways. Some measures consider only the shortest path between the two concepts while others scale this distance by the depth of the concepts in the hierarchy [120]. The former approach is simple and successful in measuring the conceptual distance between two concepts within the subsumption hierarchy of concepts. Its success even more rationalized within a domain because of the relative homogeneity of the concepts [131]. However, the proponent of the later approach argue that sibling concepts deeper in a hierarchy appear to be more closely related to one another than those higher up [128, 132]. Consequently, to take the advantages of both measures, our similarity assessment is based on a composed semantic path-based measure that considers both the local homogeneity as well as granularity of the concepts in the WordNet hierarchy of

concepts. More precisely, the composed measure we use is the *mean* of two of the path-based measures supported by the WordNet: *Path Length* and *Wu & Palmer*, where:

***Path Length* (PL)** between two concepts c_1 and c_2 is defined as the inverse of the shortest path between the synsets of the two concepts.

***Wu & Palmer* (WuP)** between two concepts c_1 and c_2 is defined as: $\frac{2 \times \text{len}(\text{root}, c_3)}{\text{len}(c_1, c_3) + \text{len}(c_2, c_3) + 2 \times \text{len}(\text{root}, c_3)}$; where c_3 is the Least Common Subsummer (LCS) of c_1 and c_2 in the hierarchy of concepts. Thus, our composed semantic path-based similarity measure of two concepts, c_1 and c_2 , can be defined as follows:

$$SSC(c_1, c_2) = (PL(c_1, c_2) + WuP(c_1, c_2)) / 2 \quad (1)$$

The correlation coefficient between the two path based measures is shown in Table 9.

To avoid repetition, the following facts and definitions are applied in all the similarity equations and functions presented in this section and the following section.

- The sum of all the weights presented in any equation is 1.
- The terms “similarity metric” or “similarity function” are exchange-ably used.
- All the similarity functions that find the similarity between two sets of elements (classes, attributes, methods, etc) are injective (see definition 5.2).
- All the similarity metrics’ values, computed in all the equations, are within the interval [0..1].

Table 9. Pearson Correlation Between Path Length and WuP Semantic Similarity Measures

Correlation coefficient	Number of compared word-pairs	P-value
0.78	20000	<0.0001

Definition 5.1: Let A and B be two sets that we want to evaluate the similarity of their elements^{*}. Let n and m be the number of elements in A and B , respectively. The *pair-wise Element Similarity matrix* (ES) of A and B is a matrix of size $n \times m$, where ES_{ij} represents the similarity score between a_i and b_j ; where $a_i \in A$ and $b_j \in B$.

Definition 5.2: Let A and B be two sets that we want to find the best match between their elements; let n and m be the number of elements in A and B , respectively, $n \leq m$. Let f be a mapping function from A to B . The mapping function f is said to be *injective* if it matches each element in A to a distinct element in B . Symbolically,

$$\forall a, b \in A, f(a) = f(b) \Rightarrow a = b.$$

5.3.1 Lexical Name Similarity Metric (NS)

Lexical Name Similarity metric (NS) measures the similarity between the names of two classes, C_1 and C_2 , based on their semantic similarity as quantified by Equation (1):

$$NS(C_1, C_2) = SSC(Name(C_1), Name(C_2)) \quad (2)$$

5.3.2 Attributes' Similarity Metric ($ASim$)

Attributes' list Similarity metric ($ASim$) measures the similarity between two sets of attributes, A_1 and A_2 , of two classes C_1 and C_2 , respectively, as follows:

$$ASim(C_1, C_2) = MAX[\forall \sum_{k=1}^{|A_1|} aSim(a_k, a_l)] / |A_2| \quad (3)$$

^{*} Elements can be classes, methods, attributes, relationships, etc.

where $a_k \in A_1$ and $a_l \in A_2$, $|A_1| \leq |A_2|$. The similarity metric $aSim(a_k, a_l)$ between two attributes is computed as a weighted similarity of their names' similarity ($NSim$) and their data types' similarity ($DTSim$):

$$aSim(a_k, a_l) = w_n \times NSim(a_k, a_l) + w_t \times DTSim(a_k, a_l), \quad (4)$$

where w_n and w_t are weights assigned to the name similarity ($NSim$), and the data type similarity ($DTSim$) respectively.

As mentioned earlier, similarity between attributes' names is computed based on their semantic similarity according to the WordNet *is-a* hierarchy of concepts, Equation (1). However, the similarity between the data type of two attributes is computed as follows:

- If the compared data types are primitive data types their similarity is the reciprocal of the shortest path between the two types according to the data type taxonomy shown in Figure 5, which is adopted from [133] .
- If the compared data types are non-primitive data types their similarity is computed according to Equation 1.
- If one of the compared data types is primitive data type while the other one is non-primitive data type they are considered as totally dissimilar and hence their similarity is 0.

When computing the similarity between two primitive data types, the shortest path is counted as the number of nodes between the two compared nodes, including the compared nodes. More precisely the similarity between two primitive data types can be computed as follows.

$$DTSim(a_k, a_l) = \frac{1}{\text{Number of nodes between Type}(a_k) \text{ and Type}(a_l)}. \quad (5)$$

For example, referring to Figure 5, the similarity between “Char” data type and “Byte” data type is 1/3 while the similarity between “Integer” data type and “Byte” data type is 1/4.

5.3.3 Operations’ Similarity Metric (*OSim*)

Operations’ list Similarity metric (*OSim*) measures the similarity between two lists of operations, O_1 and O_2 , of two classes C_1 and C_2 , respectively, as follows:

$$OSim(C_1, C_2) = \text{Max}[\forall \sum_{k=1}^{|O_1|} oSim(o_k, o_l)]/|O_2|, \quad (6)$$

where $o_k \in O_1$ and $o_l \in O_2$, $|O_1| \leq |O_2|$.

The similarity metric $oSim(o_k, o_l)$ between two operations o_k , and o_l is computed as a weighted similarity of their names’ similarity (*NSim*), parameters’ list similarity (*PLSim*), and their return type similarity (*RTSim*):

$$oSim(o_k, o_l) = w_n \times NSim(o_k, o_l) + w_{pl} \times PLSim(o_k, o_l) + w_{rt} \times RTSim(o_k, o_l), \quad (7)$$

where w_n , w_{pl} , and w_{rt} are weights assigned to the method’s name similarity (*NSim*), parameter list similarity (*PLSim*), and returned type similarity (*RTSim*), respectively.

*The parameter list similarity function (*PLSim*)* computes the similarity between two lists of parameters PL_1 and PL_2 of two methods o_1 and o_2 , respectively, as follows:

$$PLSim(o_1, o_2) = \text{Max}[\forall \sum_{k=1}^{|PL_1|} pSim(p_k, p_l)]/|PL_2|, \quad (8)$$

where $p_k \in PL_1$ and $p_l \in PL_2$, $|PL_1| \leq |PL_2|$. The similarity metric $pSim(p_k, p_l)$ between two parameters is computed in the same way $aSim$ is computed (Section 5.3.2), i.e. as a

weighted similarity of parameters' name similarity ($NSim$) and their data types' similarity ($DTSim$).

5.3.4 Internal Similarity Metric (IS)

Internal Similarity metric (IS) measures the internal similarity of two classes as a weighted similarity of their attributes' and their operations' similarity.

$$IS(C_1, C_2) = w_a \times ASim(C_1, C_2) + w_o \times OSim(C_1, C_2), \quad (9)$$

where w_a and w_o represent weights assigned to the attributes and operations similarity, respectively.

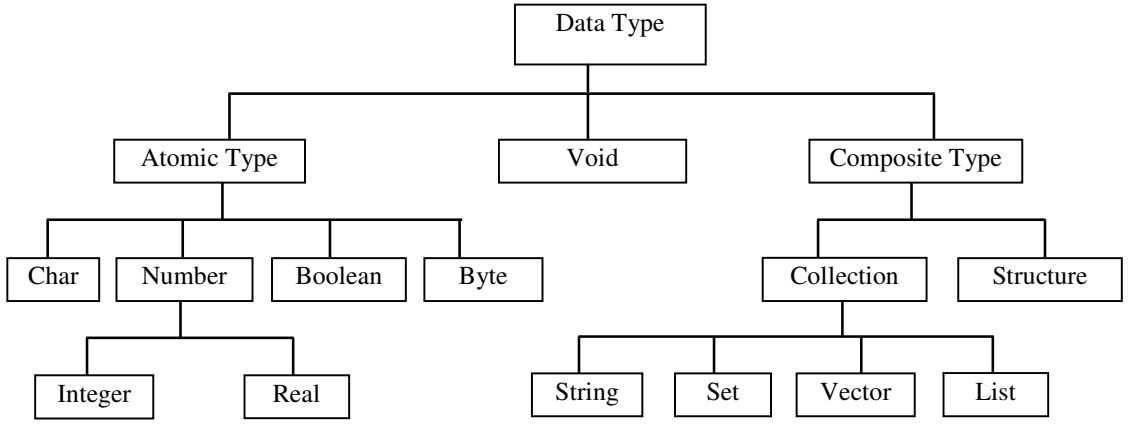


Figure 5. Data Type Taxonomy

5.3.5 Neighborhood Similarity Metric (NHS)

Neighborhood Similarity metric (NHS) measures the neighborhood similarity of two classes, C_1 and C_2 , having two sets of neighbors N_1 and N_2 , respectively, as follows:

$$NHS(C_1, C_2) = \text{Max}[\forall \sum_{k=1}^{|N_1|} NSim(n_k, n_l)] / |N_2|, \quad (10)$$

where $n_k \in N_1$ and $n_l \in N_2$, $|N_1| \leq |N_2|$.

The neighbor similarity $NSim(n_k, n_l)$ between two neighbors n_k and n_l is measured as a weighted similarity of the relation type similarity ($rtSim$), the relation name similarity ($rnSim$), and the neighbor name similarity ($nnSim$):

$$NSim(n_k, n_l) = w_{nn} \times nnSim(n_k, n_l) + w_{rn} \times rnSim(n_k, n_l) + w_{rt} \times rtSim(n_k, n_l), \quad (11)$$

where w_{nn} represents the weight assigned to neighbor name similarity, w_{rn} represents the weight assigned to relationship name similarity, w_{rt} represents the weight assigned to relationship type similarity. As mentioned earlier (Section 5.2), the neighbor name similarity and the relationship name similarity are evaluated based on the Wordnet semantic similarity while the relation type similarity is evaluated based on the similarity scores shown in Table 10, which is inspired from [45]. When evaluating the relation type similarity we consider the similarity of the two ends of the relation.

Table 10. Lookup Table of Similarities between Relationships' Ends in Class Diagram

		Relationship's End												
Relationship's End		OAS	MAS	OAG	OCO	GES	GEC	IRS	IRC	DES	DEC	RES	REC	NRE
	OAS	1	0	0.89	0.89	0	0.55	0	0.33	0	0.55	0	0.23	0
	MAS	0	1	0	0	0.55	0	0.33	0	0.55	0	0.23	0	0
	OAG	0.89	0	1	0.89	0	0.55	0	0.33	0	0.55	0	0.23	0
	OCO	0.89	0	0.89	1	0	0.55	0	0.33	0	0.55	0	0.23	0
	GES	0	0.51	0	0	1	0	0.4	0	0.72	0	0.4	0	0
	GEC	0.51	0	0.51	0.51	0	1	0	0.4	0	0.72	0	0.4	0
	IRS	0	0	0	0	0.21	0	1	0	0.49	0	0.83	0	0
	IRC	0	0	0	0	0	0.21	0	1	0	0.49	0	0.83	0
	DES	0	0.51	0	0	0.72	0	0.68	0	1	0	0.79	0	0
	DEC	0.51	0	0.51	0.51	0	0.72	0	0.68	0	1	0	0.79	0
	RES	0	0.17	0	0	0.38	0	0.89	0	0.66	0	1	0	0
	REC	0.17	0	0.17	0.17	0	0.38	0	0.89	0.66	0	0	1	0
	NRE	0	0	0	0	0	0	0	0	0	0	0	0	1
OAS = Owned Association; MAS = Member Association; OAG = Owned Aggregation; OCO = Owned Composition; GES = Generalization Supplier; GEC = Generalization Client; IRS = Interface Realization Supplier; IRC = Interface Realization Client; DES =Dependency Supplier; DEC = Dependency Client; RES = Realization Supplier; REC = Realization Client; NRE = No Relation End														

5.4 Class-to-Class Similarity

Based on the metrics presented in Section 5.3, we investigate the computation of the class similarity using seven similarity metrics: *NS*, *IS*, *NHS*, *NIS*, *NNHS*, *INHS*, and *NINHS*. The first three are, respectively, defined in equations (2), (9), and (10). The last four are defined as follows.

$$NIS(C_1, C_2) = w_n \times NS(C_1, C_2) + w_i \times IS(C_1, C_2), \quad (12)$$

$$NNHS(C_1, C_2) = w_n \times NS(C_1, C_2) + w_{nh} \times NHS(C_1, C_2), \quad (13)$$

$$INHS(C_1, C_2) = w_i \times IS(C_1, C_2) + w_{nh} \times NHS(C_1, C_2), \quad (14)$$

$$NINHS(C_1, C_2) = w_n \times NS(C_1, C_2) + w_i \times IS(C_1, C_2) + w_{nh} \times NHS(C_1, C_2), \quad (15)$$

where, w_n , w_i and w_{nh} are weights assigned to Name Similarity (*NS*), Internal Similarity (*IS*), and Neighborhood Similarity (*NHS*), respectively.

Table 11. Weight Settings of the Compound Metrics

Equation	Weight assignment	How?
Eq. 4	Evenly	Arbitrarily
Eq. 7	Arbitrary.	Arbitrarily. $w_n=0.5$; $w_{pl}=0.30$; $w_{rt}=0.20$.
Eq. 9	Adopted	Based on the complexity [124]. $W_a=0.4$; $w_o=0.6$;
Eq. 11	Calibrated	Experimentally, see Section 8.5
Eq. 12	Calibrated	Experimentally, see Section 8.5
Eq. 13	Calibrated	Experimentally, see Section 8.5
Eq. 14	Calibrated	Experimentally, see Section 8.5
Eq. 15	Experimentally	Experimentally, see Section 8.5

Having a compound similarity metric as a combination of different other metrics entails that each metric in the combination should be assigned a weight that allows for an

accurate similarity/dissimilarity assessment. Table 11 summarize the weight setting in the different similarity. For the weights that are set experimentally, empirical experiments and analysis are provided in Section 8.5.

5.5 Tool Support for Metrics Collection

We developed a Java-based tool that takes, as input, a set of class diagrams in XMI format. The tool then parses the XMI files to extract the required similarity information for the similarity metrics, and then the tool assesses the pair-wise similarity between the classes of each pair of input models based on the different types of similarity metrics. For each similarity metric used, the pair-wise similarity scores between the classes of each pair of models is presented by the tool as a two dimensional similarity matrix, *ES*.

5.6 Summary

In this chapter we presented the similarity assessment framework in terms of the similarity aspects and similarity metrics used for assessing the class diagrams similarity. The focus of the chapter was to discuss the different similarity aspects and how to handle them to improve the similarity assessment framework. The chapter also introduced formal definitions for all the metrics used in our comparison framework. Additionally the chapter presented the weight setting schemes for the compound metrics. Empirical validation and analysis related to the comparison framework are presented in Section 8.5 of Chapter 8.

CHAPTER 6

MODEL MATCHING

6.1 Introduction

Within the context of our framework we define model matching between a pair of two models as the task of mapping each element in the smaller model of the pair (model with fewer number of classes) into its most similar element in the other model, given the similarity scores between the elements of the two models as quantified by the similarity metrics. Accurate similarity assessment (comparison) leads to accurate matching, and accurate matching leads to a duplication-free merging [58].

Model matching task is time consuming due to the fact that finding the optimal match between the elements of two models is a kind of combinatorial problem generally referred to as graph matching problem [59]. Therefore, an efficient matching algorithm is required to obviate the complexity of the brute-force method and meanwhile provide an acceptable solution. One of the approaches is to make some plausible assumptions which can be driven by utilizing the characteristics of the problem in hand. An alternative way is to go with some heuristic based solutions.

It is crucial for an effective and efficient matching to have efficient matching algorithms as well as good similarity metrics for quantifying the similarities of the models to be matched. In the previous chapter (Chapter 5) we discussed different similarity aspects and the different factors that lead to better similarity assessment between the elements of the compared models. Our focus in this chapter is the matching task and how to tackle its complexity. In other words, the chapter is centered around

different matching algorithms. The input to any matching algorithm is the pair-wise similarity scores between the elements of the matched models, represented in ES matrix.

As mentioned in Chapter 4, a 3-stage matching mechanism have been proposed to tackle the complexity of the matching. These stages are technically detailed in Sections 6.2, 6.3, and 6.4.

6.2 First Stage: Element-to-Element Matching

Definition 6.1: Let M_1 and M_2 be two models, with n and m classes, respectively, where $n \leq m$; let ES be the pair-wise element similarity matrix of M_1 and M_2 . The optimal injective match is an injective match from M_1 to M_2 where each element in M_1 is matched to a distinct element in M_2 with which it is the most similar.

Definitions 5.1 and 5.2 are also necessary for the presentation of the first stage matching. The focus of this stage of our framework is to look for an optimal injective match between each pair of models, given their ES matrix as an input. If we have a pair of models M_i and M_j of n and m classes, respectively, with $n \leq m$, the brute-force algorithm to find the optimal match entails finding all possible injective matches between M_i and M_j . Then the injective match with the highest similarity value is retained. However, this requires exploring $n!$ possible injective matches, resulting in an exponential time complexity.

A trivial solution is to go with a simple greedy approach. Given a sequence of row indices (representing classes of M_i) of ES matrix, the simple greedy matching algorithm (SGRM), Figure 6, goes over the sequence row by row, matching each row element of M_i to a column element of M_j with which it has the highest similarity score, $ES_{i,j}$; if the

algorithm finds the column element corresponding to the highest similarity score as already matched, it looks for the next highest available. This simple algorithm can find an injective match between a pair of models. However, this match is not guaranteed to be optimal. The sequence in which the rows are visited by the algorithm is a major factor in getting the optimal match. For example, let Figure 7-(a) depict the similarity scores between the elements of any two models, M_1 (rows' elements in the matrix) and M_2 (columns' elements in the matrix). Assume, the algorithm visits the rows in an increasing order of row indices, i.e. row 0, then row 1, and so on, until row 6, matching the elements in a greedy-like manner. In particular, when row 0 (which represents the similarity scores between the class c_0^1 of M_1 and each class c_j^2 of M_2) is visited before row 2, the algorithm matches the class c_0^1 to the class c_3^2 , marking both c_0^1 and c_3^2 as matched classes with a similarity score of 0.56. Then, when visiting row 2, c_3^2 is found to be most similar to c_2^1 , $ES_{2,3} = 0.91$. However, c_3^2 is already matched with c_0^1 , with $ES_{0,3} = 0.56$, and thus cannot be re-matched with c_2^1 . This is clearly an indication of wrong match, and it results in another wrong match, as the algorithm will, enforcedly, matches c_2^1 to c_4^2 , which in turn causes a third wrong match between c_3^1 and c_6^2 , as c_3^1 would be best matched to c_4^2 . However, if the algorithm, during its execution, followed the sequence of rows 1, 2, 3, 4, 6, 0, 5, Figure 7-(b), the optimal match would be obtained. The matching similarity matrix corresponding to the ES matrix depicted in Figure 7-(a) is shown in Figure 8.

As we have seen in the aforementioned demonstration, the SGRM algorithm lacks the global view of the solution space. This short insight of the algorithm comes from the fact that when the algorithm matches the elements of the two models, it cannot go beyond the

horizon of a single row. If the algorithm had a global view about the ES matrix when matching c_3^2 , it would not match c_3^2 with c_0^1 .

Algorithm SGRM: Simple Greedy Matching Algorithm

Input: two dimensional matrix $ES[n][m]$ where $ES[i][j]$ represents the similarity score between class C_i^1 of model M1 and class C_j^2 of model M2, with n and m represent the number of classes in M1 and M2, respectively;

An sequence S of distinct integer numbers representing the row indices in ES Matrix, $|S| = \min\{m, n\}$.

Output: a two dimensional matrix $MSM[3][\min\{m, n\}]$, such that $MSM[0][j]$ and $MSM[1][j]$ represent the indices of j^{th} matched pair of similar classes from M1 and M2 respectively, with $MSM[2][j]$ represent their similarity score.

1. for $i \leftarrow 1$ to $|S|$ do
 2. find $ES_{S_i, k}$, where $ES_{S_i, k} = \max_{j=0,1,\dots,m} ES_{S_i, j}$ such that $C_{S_i}^1$ and C_j^2 are not matched
 3. Mark $C_{S_i}^1$ and C_k^2 as matched.
 4. end for
 5. return MSM
-

Figure 6. SGRM Algorithm

	c_0^2	c_1^2	c_2^2	c_3^2	c_4^2	c_5^2	c_6^2	c_7^2		c_0^2	c_1^2	c_2^2	c_3^2	c_4^2	c_5^2	c_6^2	c_7^2
c_0^1	0.36	0.29	0.46	0.56	0.44	0.54	0.45	0.35	c_0^1	0.36	0.29	0.46	0.56	0.44	0.54	0.45	0.35
c_1^1	1	0.26	0.32	0.42	0.34	0.34	0.32	0.33	c_1^1	1	0.26	0.32	0.42	0.34	0.34	0.32	0.33
c_2^1	0.28	0.21	0.32	0.91	0.60	0.34	0.34	0.25	c_2^1	0.28	0.21	0.32	0.91	0.60	0.34	0.34	0.25
c_3^1	0.34	0.28	0.51	0.58	0.89	0.46	0.62	0.39	c_3^1	0.34	0.28	0.51	0.58	0.89	0.46	0.62	0.39
c_4^1	0.34	0.27	0.5	0.54	0.46	0.90	0.4	0.68	c_4^1	0.34	0.27	0.5	0.54	0.46	0.90	0.4	0.68
c_5^1	0.44	0.3	0.44	0.53	0.57	0.48	0.40	0.41	c_5^1	0.44	0.3	0.44	0.53	0.57	0.48	0.40	0.41
c_6^1	0.4	0.67	0.63	0.56	0.42	0.4	0.4	0.84	c_6^1	0.4	0.67	0.63	0.56	0.42	0.4	0.4	0.84
(a) Row sequence resulting in the shaded match is: 0, 1, 2, 3, 4, 5, 6									(b) Row sequence resulting in the shaded match is: 1, 2, 3, 4, 6, 0, 5								

*Shaded numbers represent the similarity scores of matched elements;

*Numbers in **bold** represent the max in the row; when every bolded number is shaded, the match is optimal with zero error.

Figure 7. Pair-wise Element Similarity Matrix Between Classes of Two Models, M1 and M2

M_1 classes	c_0^1	c_1^1	c_2^1	c_3^1	c_4^1	c_5^1	c_6^1	Over all Similarity between M_1 and M_2 4.96/7 = 0.71
M_2 classes	c_3^2	c_0^2	c_4^2	c_6^2	c_5^2	c_2^2	c_7^2	
Sim. Score	0.56	1	0.60	0.62	0.90	0.44	0.84	

Figure 8. Matching Similarity Matrix between Classes of Two Models, SGRM Algorithm

In this work, we propose three polynomial time algorithms for model matching: a Global Greedy algorithm (GGRM), detailed in Section 6.2.1, a hybridized Greedy-Genetic algorithm (GGAM), detailed in section 6.2.2 and a hybridized Greedy-Simulated-Annealing algorithm (GSAM), detailed in section 6.2.3 .

6.2.1 Proposed Greedy Matching Algorithm (GGRM)

The intuitive and the plausible assumption underlying this algorithm is that a pair of classes with the highest similarity values represents the most relevant classes. Following this assumption the GGRM algorithm, should find the optimal match between the classes of two models in a polynomial time. The steps of the algorithm are listed in Figure 9 and exemplified in Figure 10. Given the pair-wise element similarity matrix, ES , between the classes of any two models, M_1 and M_2 , the algorithm, in each of its steps, looks for the highest similarity score, in the ES matrix, for which the corresponding classes are not matched so far. Then the algorithm matches these classes and marks them as matched. The algorithm repeats its steps until all the classes of the smaller model (i.e. model with less number of classes) are matched.

Algorithm GGRM: Proposed Greedy Matching Algorithm

Input: two dimensional matrix $ES[n][m]$, where $ES[i][j]$ represents the similarity score between class C_i^1 of model M1 and class C_j^2 of model M2, with n and m are the number of classes in M1 and M2, respectively, and $n \leq m$;

Output: two dimensional matrix $MSM[3][n]$, such that $MSM[0][j]$ and $MSM[1][j]$ represent the indices of j^{th} matched pair of similar classes from M1 and M2 respectively, with $MSM[2][j]$ represents their similarity score.

1. While there is unmatched class in M1 do
 2. find $ES_{k,l}$, $ES_{k,l} = \max_{i=0,1,...,n, j=0,...,m} ES_{i,j}$ such that C_i^1 and C_j^2 are not matched
 3. Mark C_k^1 and C_l^2 as matched.
 4. end for
 5. return MSM
-

Figure 9. GGRM Algorithm

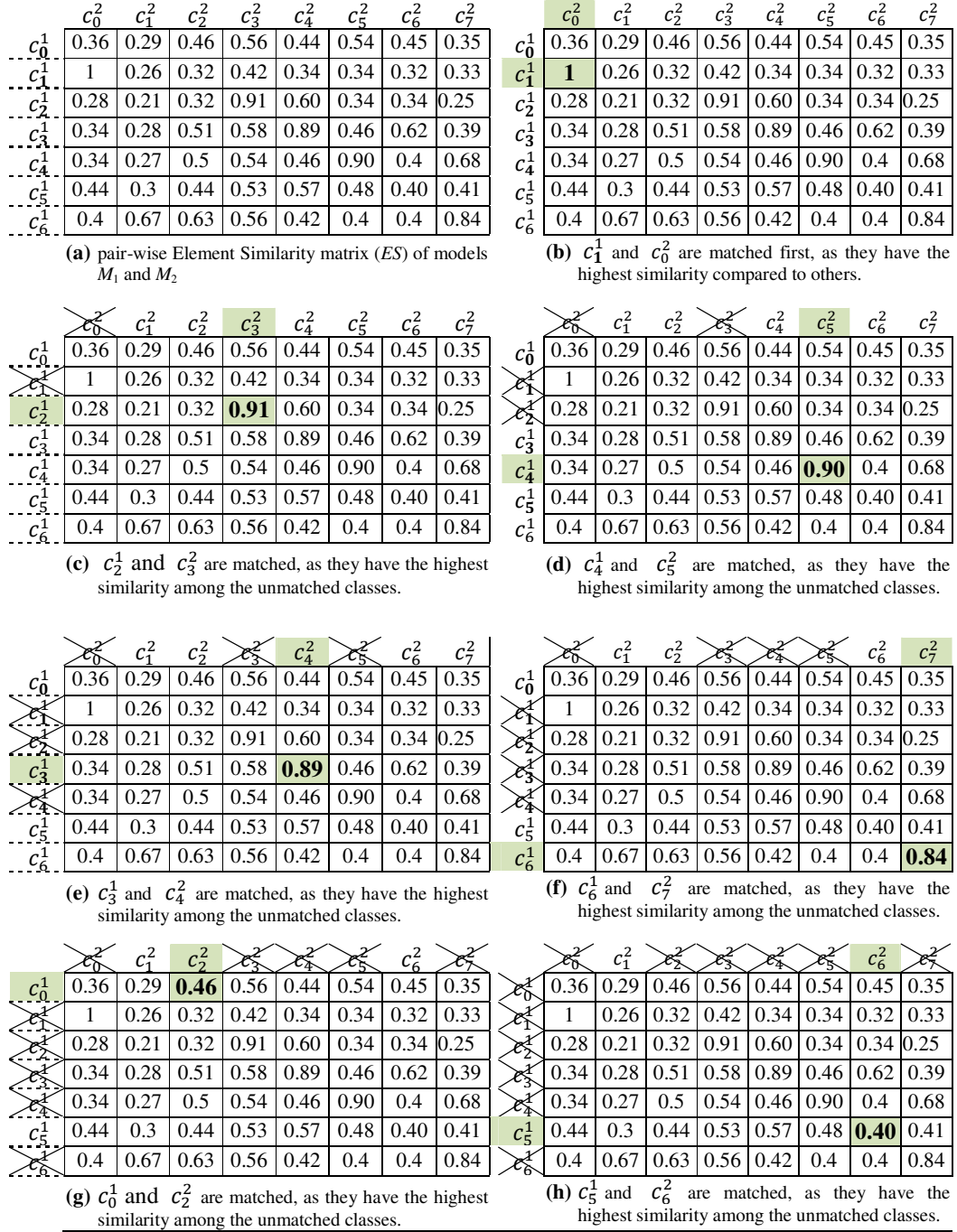


Figure 10. An Illustrative Example of the Proposed Greedy Matching Algorithm (GGRM)

Figure 11 shows the matching similarity matrix as an output of the GGRM algorithm. The time complexity of the algorithm is $O(mn^2)$ where n is the number of classes in the smaller model and m is the number of classes in the larger model.

The particularity of our matching problem is what actually makes GGRM algorithm works fine. In other words, the aforementioned assumption underlying this algorithm is intuitive when looking for the most similar pairs of elements between the two models (class level similarity). However, if our objective is to look for the maximum overall similarity between the two models (model level similarity), as it is the case with many optimization problems, e.g. job-assignment problem or travelling salesman problem, etc, the GGRM algorithm may easily get trapped in the local optima. Therefore, a matching algorithm with better global insight is needed.

Population-based techniques like Genetic Algorithms (GA) (see Section 2.8) provides a better exploration for the solutions' search space. This population-based exploration helps the algorithm to avoid being trapped in the local optima which is an intrinsic characteristic in the greedy algorithms [84]. Therefore, a Greedy-Genetic Matching algorithm (GGAM), is proposed as another matching algorithm for model matching. The use of more than one algorithm for model matching has twofold objective. First, the result of each algorithm can be validated against the other ones. Second, in some situations, the use of one of the algorithm is more rationalized than the use of the other.

M_1 classes	c_1^1	c_2^1	c_4^1	c_3^1	c_6^1	c_0^1	c_5^1	Over all Sim between
M_2 classes	c_0^2	c_3^2	c_5^2	c_4^2	c_7^2	c_2^2	c_6^2	M_1 and M_2
Sim. Score	1	0.91	0.90	0.89	0.84	0.46	0.40	$5.4/7 = 0.77$

Figure 11. Matching Similarity Matrix between Classes of Two Models, GGRM Algorithm

6.2.2 Hybridized Greedy-Genetic Matching Algorithm (GGAM)

Traditional implementation of the GA algorithm involves an intrinsic randomness, which can lead to problems in both the convergence and the performance of the algorithm. It can also lead to invalid solutions in many problems. This encouraged the

researchers to hybridize the traditional form of the GA with some ideas of other algorithms, with the objective to improve the quality and the convergence time of the algorithm as well as the correctness of the solution. The hybridization can be adopted to any building block of the algorithm. For example, in [75] some greedy ideas are adopted to improve the generation of the initial population, the crossover operation, and the mutation operation.

At any iteration during its evolution, the evolutionary algorithms, including GA, usually work on a complete and valid solution [84]. If we terminate the algorithm at any iteration we can still have a solution at hand. However, this solution may not be optimal. The evolution process towards the (near-) optimal solution depends heavily on the algorithm settings. For the GA algorithm, among these settings is the fitness function. For example, when applying the GA for the known Travelling Salesman problem, traditionally, the algorithm will generate a sequence of cities and the fitness function is the summation of the distances between these cities, following the given sequence, in order. In our matching problem, assume two models M_1 and M_2 with n_1 and n_2 number of classes, respectively, where $n_1 \geq n_2$. The typical way to implement GA is to encode the candidate solution (chromosome) as a one-dimensional array S of length n_2 , where the values of S represent the classes of M_1 , as a permutation of distinct integers, $0 \leq S[i] \leq n_1$, while the static indices, $0, 1, 2, \dots, i, \dots, n_2 - 1$, of S represent the classes of the model M_2 . The indices of S can be visualized as a static chromosome while the values of S can be visualized as a dynamic chromosome on which the GA operations are applied. The matching is then performed in such a way that a class of M_1 represented by the value $S[i]$ is matched to a class in model M_2 represented by the index i . The fitness function of a

candidate solution is defined as the summation of the similarity scores between the pairs of matched classes as quantified in the *ES* matrix, where the value $S[i]$ is an index of a row while the index i is an index of a column in the *ES* matrix. This approach has been followed in many similar problems whose solutions are in the form of permutations of integer numbers, e.g. [78, 134].

We propose an enhancement to the setting of the traditional approach as follows. Instead of mapping the pairs of classes, class $S[i]$ of M_1 to the class i of M_2 , the matching is performed in a greedy manner. Given a sequence of classes of one of the models, as represented by the GA chromosome, which in turn represent row indices of the *ES* matrix, a simple greedy algorithm (SGRM, Figure 6) goes over the sequence row by row, matching each row element of M_1 to a column element of M_2 with which it has the highest similarity score; if the algorithm finds the column element corresponding to the highest similarity score as already matched, it looks for the next highest available. This enhancement is assumed to make the algorithm converge faster to the optimal solution as it avoids the randomness involved in the traditional implementation of the algorithm when matching the elements and calculating the fitness score. The penalty is that the complexity time of the matching step becomes $O(n^2)$, instead of $O(n)$ in the traditional approach. However, this increase in the complexity can be compensated by the fast convergence of the enhanced algorithm. In other words the algorithm can converge to the (near-) optimal solution in less number of iterations as compared to the traditional approach.

Adopting the hybridized Greed-Genetic algorithm (GGAM) to the matching problem requires setting up some parameters and some adaptation of its building blocks to suite the matching problem in hand. We discuss these settings as flows.

Problem formulation: In the beginning of this section we mentioned that algorithm SGRM can find the optimal injective match if it follows an appropriate sequence of row indices. However, as we mentioned, the complexity time of finding this sequence using the brute-force approach is exponential. Therefore, our matching problem can be reformulated as a search problem with the objective of finding the appropriate sequence of row indices to be followed by SGRM, on the *ES* similarity matrix, in order to give us the optimal injective match.

Chromosome encoding: Being centered around the evolution of the chromosome, the first step in the genetic algorithm is to encode any potential solution into a form of a chromosome so that the genetic concepts can be applied to it. Since our solution is a sequence of row indices (in *ES* matrix), which, in turn, represents a sequence of indices of classes of one of the models in the pair, each chromosome needs to represent a valid sequence, in which no row index appears more than once. In the case when the two models have the same number of classes, the length of the chromosome is equal to the number of classes in any of the two models and the genes of the chromosomes represent row indices. However, when the number of classes in the two models is not the same, the length of the chromosome is equal to the number of classes in the smaller model, but the genes in the chromosome can be in one of two cases. If they represent the indices of the classes in the smaller model they are already a valid representation of a candidate solution. If, however, they represent the indices of the classes in the larger model, some indices will be truncated, as the number of indices is larger than the length of the chromosome. Since the algorithm works on a population of solutions, and because each initial solution is generated randomly, it is highly unlikely that different individuals will

miss the same index (indices). This means that, although some indices are missed out from some of the candidate solutions they will show up in other ones, indicating their existence over the generation. Moreover, the truncated indices will be kept in a pool with which the mutation operator probabilistically performs swapping.

Initial solution: The initial population of solutions is generated randomly as a sequence of integers representing the class indices in one of the pair of the matched models. We develop a special generator to guarantee that each individual is a valid solution.

Fitness function: As previously mentioned, each candidate solution represents a sequence of indices of the classes of one of the models in the pair. This sequence is given to SGRM algorithm to follow in order to find the corresponding injective match. The sum of the similarities between the matched elements in this injective match is used as fitness function, the higher the sum the fitter the solution. Selecting the number of elements passing the threshold would be another option as a fitness function, but one problem with this is that if two elements (say x and y) in one model have the similarity scores 0.8 and 1.0, respectively, with an element z of another model, then the algorithm will not differentiate between the two cases. In other words, if z is already matched with x for any intermediate solution, changing the match to become between z and y may not change the value of the fitness function (if this is the only change in the new solution), assuming that the threshold is 0.8.

Using the error, in terms of the difference between the similarity scores of the matched elements and the maximum values in the row or column, would be a third option for the fitness function. However, the problem with this measure as a fitness function is

that the algorithm will try to minimize the error rather than looking for the most similar elements. In other words, the algorithm will not differentiate between a *current* situation where a match exists between two elements with low similarity score and 0 error, and a better new match with high similarity score and 0 error. In fact it is possible to prefer the former case over the later one if it will result in minimizing the overall error. This situation is likely to happen when the number of elements in the two models are different.

Genetic operators: When applying the simple crossover and the mutation operators [135] to any of the candidate solutions to our problem they do not work well as they may result in invalid solution, i.e. some indices may be repeated while others are missed out. Therefore these two operators need to be adapted in a certain way, so that they still mimic the biological gene evolution. It is not just the uniqueness and the omission of the genes (indices), rather, the crossover is supposed to preserve previous advances in the solutions and incorporate them into future solutions [81]. On the other hand, the role of the mutation operator is to introduce diversity in the population of the solutions, which is needed to ensure an appropriate coverage of the solution space and thus prevent the premature convergence of the whole population to sub-optimal solutions.

Crossover: Two common crossover operators can be used, Partially Mapped crossover (PMX) [136] and Order crossover (OX) [84]. The PMX crossover operator builds an offspring by choosing two cut-points in the two parents, copying the subsequences between the cut points in the two parents into new two offsprings, one each, and then the remaining indices are filled, position wise, from the other parent [84, 136].

The OX operator builds offspring by choosing a subsequence of one parent and preserving the relative order of indices from the other parent. It capitalizes on the importance of the relative order of the indices rather than their specific positions [84, 137]. Guided by recommendations in [80, 84, 138], we opt to use order crossover (OX) operator. Figure 12 shows how the offspring is generated using OX operator. First, two cut-points are selected randomly in the two parents, see Figure 12-b where the cut-points are marked with dashed borders. Then, the subsequences between the cut points in the two parents are copied into new two offsprings, one each, Figure 12-c. Then, the remaining indices, starting after the second cut-point, are filled from the other parent, in order, omitting those which already exist in the copied subsequence. The crossover probability (also known as, crossover rate) p_c controls the frequency in which the crossover is applied. Too high crossover rate may result in over-exploitation of the current individuals. As a result, new areas in the search space may not get explored. A low crossover rate may delay the convergence to a promising region of the search space [139]. Typical values of p_c are in the range 0.5-1.0 [140, 141].

Mutation: Mutation is performed in two different ways. When the available sequence is larger than the length of the chromosome (this happens when the compared models are of different sizes), a pool representing the extra indices of the classes of the larger model is maintained. Then the mutation is performed by a random selection of a position in the chromosome and swapping its content probabilistically with an index selected randomly from the pool or with another position selected randomly in the chromosome. The former case (i.e. the selection from the pool) is a type of mutation generally referred to as *immigration* [75]. If, however, no indices are maintained in the pool, the mutation is

performed by swapping the contents of two positions of the chromosome, selected randomly. The mutation rate p_m controls the frequency in which the mutation is applied. High mutation rate renders the GA into random search algorithm. A very low mutation rate results in not reaching the global optima. A small mutation rate less than 0.1 is commonly recommended [140, 141].

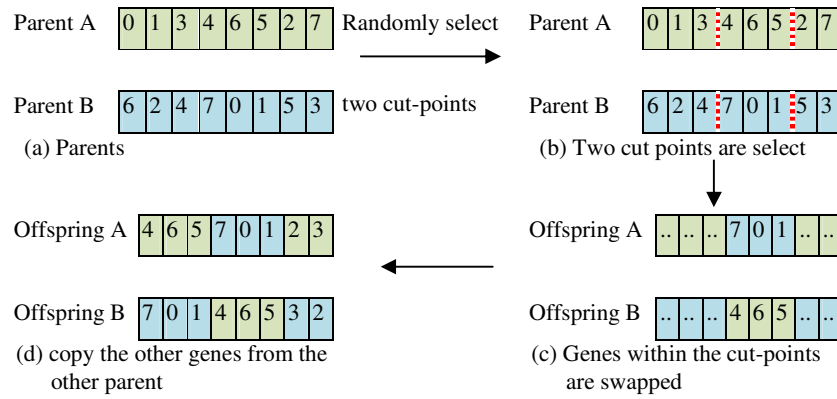


Figure 12. Genetic Crossover

To sum up, in our experiments the GA parameters are set, guided by the literature recommendations, as shown in Table 12. Section 8.6 empirically investigate the performance of the two algorithms over different problem sizes. The investigation clearly shows the effectiveness of greedy idea in speeding up the convergence of the genetic algorithm to the optimal solution.

Table 12. GA Parameters Settings

Population size	30
Number of generations	10000
Crossover rate	0.70
Mutation rate	0.10
Immigration rate	0.50 of mutation rate
Selection method	Roulette wheel Best half (50%)

6.2.3 Hybridized Greedy Simulated Annealing Matching Algorithm (GSAM)

In Section 2.9 we provided a brief background about the Simulated Annealing algorithm (SA). In this section we introduce the Greedy-Simulated Annealing algorithm (GSAM). To implement the GSAM algorithm within the context of our class diagram matching we follow the same encoding scheme used in the GGAM algorithm, where the solution is encoded as a sequence of distinct integers representing the classes in one of the matched models. This sequence is traversed by the SGRM algorithm to find an injective match with the classes of the other model. This is actually where the algorithm is hybridized with the greedy idea. The objective function is computed as the sum of the similarity scores as quantified in the *ES* matrix between the corresponding matched elements of the two models. Solution which leads to higher similarity score than the current one is always accepted. Worse solution is accepted probabilistically.

Using SA requires setting up some parameters, such as the cooling rate, initial temperature, as well as defining the objective function so that the quality of the different solutions can be compared. In our experiment the GSAM environment was set as follows. The initial solution is generated as a random sequence of distinct integers representing row elements' indices in the *ES* matrix. Initial temperature (T), cooling rate, and termination condition are set, guided by some recommendations in the literature [142, 143], into 1000, 0.01, 0.1, respectively. The acceptance probability (P) is calculated as follows. $P = \exp(-F(S_{\text{new}}) - F(S_{\text{current}})) / T$, where $F(S_{\text{new}})$ is the objective function value of the new solution; $F(S_{\text{current}})$ is the objective function value of the current solution. Neighbor solution is generated by swapping the contents of two randomly selected

locations in current solution. The objective function is computed as the sum of the similarity scores of the matched pairs of classes between the two models.

6.2.4 Summary of the First Stage Matching

The first stage matching algorithms emphasize on element to element (or one-to-one) matching between the classes of the two matched models. The final optimal match may contain elements that are matched just due to the injectivity property of the mapping function despite their low similarity scores. These elements will be filtered out by the first stage matching threshold filter.

6.3 Second Stage: One-to-Many Matching

Elements not passing the one-to-one similarity threshold, in the first stage, needs to be further investigated for potential similarity through more complex similarity assessment that can capture some of the design differences that was not able to be captured by the first stage's similarity assessment mechanism. The following definitions are necessary for the presentation of the second stage algorithm.

Definition 6.2: A Class-group I , in a model M_k , denoted as CG_I^k , represents those set of classes in M_k modeling the same domain concept, where I represents the set of indices indexing those classes in model M_k .

For example, in Figure 1 the two classes “Scheduled Flight” and “Offered Flight”, of M_0 are representing the same concept which is modeled as a single class (Flight) in M_2 . Therefore, the two classes, “Scheduled Flight” and “Offered Flight”, represent one class-group.

Algorithm GMA: Group-Matching Algorithm

Input: Two fragments F1 and F2 as subsets of two different models M1 and M2 consisting of n and m number of classes, respectively, with $c_i^1 \in F1 \subseteq M1$ and $c_j^2 \in F2 \subseteq M2$.

Output: A two dimensional matrix GSM[4][N-1], representing the best match between the class-groups of M1 and M2,

```
1. for i ← 1 to n do
2.   for j ← 1 to m do
3.     IS[i][j] ← findInternalSim( $c_i^1$ ,  $c_j^2$ )
4.     NHS[i][j] ← findNeighborhoodSim( $c_i^1$ ,  $c_j^2$ )
5.     INHS[i][j] ←  $w_i \times IS[i][j] + w_{nh} \times NHS[i][j]$ 
6.   end for
7. end for
8. done ← false
9. while not done do
10.  //go row wise as follows
11.  For each  $c_i^1$  in F1
12.    find the most similar class  $c_k^2$  in F2
13.    let yGroup = { $c_k^2$ } and xGroup = { $c_i^1$ }
14.    let simSofar = INHS[i][k]
15.    for each  $c_j^2$  in F2,  $j \neq k$ 
16.      if ( $c_j^2$  is a neighbor of elements in yGroup and adding  $c_j^2$  to yGroup will
        improve its similarity with xGroup) then
17.        yGroup ← yGroup  $\cup$  { $c_j^2$ };
18.        update simSofar
19.      end if
20.    end for
21.    rowWiseSim[0][i] ← simSofar; rowWiseSim[1][i] ← i;
22.  end for
23.  //go column wise as follows
24.  foreach  $c_j^2$  in F2
25.    find the most similar class  $c_k^1$  in F1
26.    let yGroup = { $c_j^2$ } and xGroup = { $c_k^1$ }
27.    let simSofar = INHS[k][j]
28.    foreach  $c_i^1$  in F1,  $i \neq k$ 
29.      if ( $c_i^1$  is a neighbor of elements in xGroup and adding  $c_i^1$  to xGroup will
        improve its similarity with yGroup) then
30.        xGroup ← xGroup  $\cup$  { $c_i^1$ };
31.        update simSofar
32.      end if
33.    end for
34.    colWiseSim[0][j] ← simSofar; colWiseSim[1][j] ← j;
35.  end for
36.  Sort(rowWiseSim, descending);
37.  Sort(colWiseSim, descending)
38.  if (rowWiseSim[0][0] ≥ colWiseSim[0][0] and rowWiseSim[0][0] ≥ threshold) then
39.    Mark  $c_{rowWiseSim[1][0]}^1$  and the corresponding yGroup as matched classes, add them to
    MSM, and remove them from F1 and F2, respectively.
40.  elseif (colWiseSim[0][0] > rowWiseSim[0][0] and colWiseSim[0][0] ≥ threshold) then
41.    Mark  $c_{colWiseSim[1][0]}^2$  and the corresponding xGroup as matched classes, add them to
    MSM, and remove them from F2 and F1, respectively.
42.  Else done=true
43.  End if
44. End while
45. Return MSM
```

Figure 13. Second Stage Matching Algorithm

Definition 6.3: Let CG_I^k be a class-group in a model M_k , the neighborhood (NH) of CG_I^k in model M_k is defined as the set of classes that have a direct relationship with any class of CG_I^k :

$NH(CG_I^k) = \{c_j^k : c_j^k \in M_k \text{ and } c_j^k \text{ has direct relationship with any class in } CG_I^k\}$.

Definition 6.4: Let CG_I^k be a class-group in a model M_k , the list of attributes (A) of CG_I^k are defined as the collection of the attributes in all the classes involved in the class-group CG_I^k :

$A(CG_I^k) = \{a : a \in C_i, C_i \text{ is a class in } CG_I^k\}$.

Definition 6.5: Let CG_I^k be a class-group in a model M_k , the list operations (O) of CG_I^k are defined as the collection of the operations in all the classes involved in the class-group CG_I^k :

$O(CG_I^k) = \{o : o \in C_i, C_i \text{ is a class in } CG_I^k\}$.

In this stage a single element from certain model (say M_i) can be matched to more than one element in another model (say M_j) based on a weighted combination of both internal and neighborhood similarity values. In particular, let F_1 and F_2 be two subsets of classes not passing the first stage similarity threshold and of size n and m , respectively. Let $C_i^1 \in F_1 \subseteq M_1$ and $C_j^2 \in F_2 \subseteq M_2$, where M_1 and M_2 are two models consisting of n_1 and n_2 classes respectively. The algorithm first finds the similarity between each class C_i^1 from F_1 (row elements) and each class C_j^2 from F_2 (column elements). Then, the algorithm proceeds as follows. First it goes row-wise, starting at row 0, looking for C_j^2 that has the maximum similarity score with C_0^1 . This maximum similarity between C_0^1 and C_j^2 is considered as the best similarity so far, and thus a new class-group called $CG_{\{j\}}^2$ is created with C_j^2 is the first class in the class-group. The algorithm then tries to add the

other classes C_k^2 , where $k \neq j$, to $CG_{\{j\}}^2$ one at a time, evaluating the similarity between C_0^1 and $CG_{\{j,k\}}^2$ after adding C_k^2 ; if the similarity is improved the class C_k^2 is included in the class-group ($CG_{\{j,k\}}^2$) and the similarity so far is updated; otherwise the class C_k^2 is excluded from the class-group ($CG_{\{j\}}^2$), trying another class. The algorithm then proceeds with the other rows in the same way, looking for C_j^2 that has the maximum similarity score with C_i^1 , creating a new class-group $CG_{\{j\}}^2$ with a class C_j^2 being the first class in the class-group, adding to $CG_{\{j\}}^2$ those classes that improve the similarity against C_i^1 , and updating the similarity scores accordingly. The similarity between each class C_i^1 from $F1$ and the corresponding group CG_i^2 of classes from $F2$ is saved in an array (called *rowWiseSim*), sorted in descending order according to the similarity scores.

The algorithm then goes column-wise, in the same manner, starting at column 0, looking for C_i^1 that has the maximum similarity score with C_0^2 . This maximum similarity between C_0^2 and C_i^1 is considered as the best similarity so far, and the class C_i^1 is added as the first class in a class-group $CG_{\{i\}}^1$. The algorithm proceeds in the same way with the other columns. The similarity between each class C_j^2 from $F2$ and the corresponding group of classes CG_j^1 from $F1$ is saved in an array (called *colWiseSim*), sorted in descending order according to the similarity scores. Since the two arrays are sorted in descending order, the maximum similarity in the two arrays will be either in *rowWiseSim*[0] or in *colWiseSim*[0]. If the maximum is in *rowWiseSim* [0] and this maximum satisfies the second stage threshold, the corresponding C_i^1 and CG_i^2 are marked

as matched class-groups^{*} and added to the similarity matrix of the matched classes of models M_1 and M_2 . Similarly, if the maximum is in $colWiseSim[0]$ and this maximum satisfies the second stage similarity threshold, the corresponding C_j^2 and CG_j^1 are marked as matched class-groups and added to the similarity matrix of the matched elements of models M_1 and M_2 . The matched classes are removed from further consideration. The algorithm repeats its steps until no further possible match. It is worth mentioning here that for two classes to be combined, they must be adjacent to each other.

6.4 Third Stage: Residual Matching

In the second stage algorithm (Section 6.3) the focus is on the situation where a single class in one model can be modeled as multiple classes in the other model, as they are representing the same underlying concept. However, we may have a situation where the same underlying concept can be modeled as multiple classes in the two models considered in matching. This situation cannot be captured by the second stage matching algorithm. Therefore we propose a third stage algorithm to handle such a situation. This stage is an extension of the second stage. If we denote by R_1 and R_2 the set of residual classes not passing the first and the second similarity threshold, where $R_1 \subseteq M_1$ and $R_2 \subseteq M_2$, then the algorithm just improves the similarity of the class-groups formed in the second stage by adding each class in R_1 or R_2 to the most suitable class-group, if possible, based on the contribution of the added class to the similarity improvement between the corresponding matched class-groups. In other words, if we designate $\Delta_{i,j}^k$ as the similarity improvement achieved when adding class C_i^1 (where $C_i^1 \in R_1$) to the class-group CG_k^1

^{*} When a single class is matched against a class-group we consider this class as a class-group of a single class.

and/or the class C_j^2 (where $C_j^2 \in R_2$) to the class-group CG_k^2 , then the algorithm aims at improving the similarity of the matched class-groups CG_k^1 and CG_k^2 by adding the classes C_i^1 and/or C_j^2 to CG_k^1 and/or CG_k^2 , respectively, where $\Delta_{i,j}^k$ is maximum.

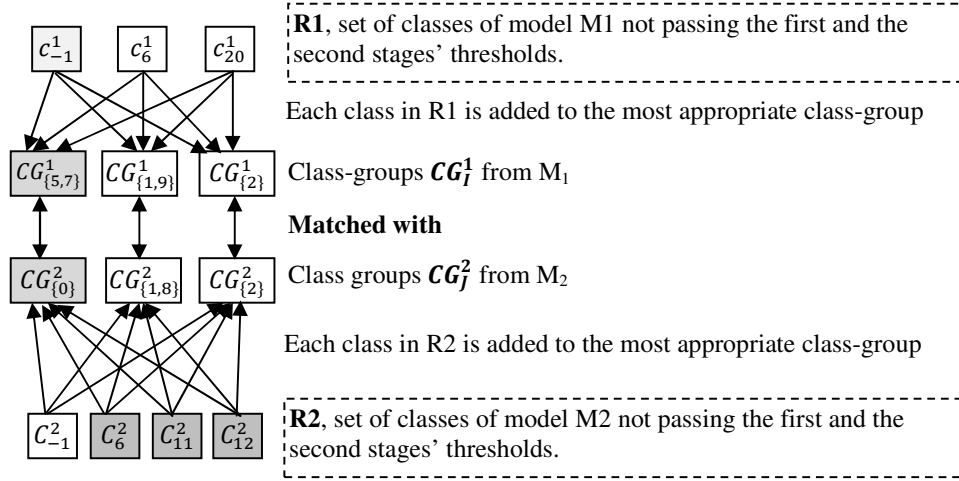


Figure 14. An Illustrative Example of the Steps of the Third Stage Matching Algorithm

Figure 14, shows an illustrative example about how the third stage matching is performed. Classes and class-groups are represented in boxes. The superscript represents the model index to which the class or the class-group belongs. The subscript represents the index of the class or the class-group within the model indexed by the superscript. When the class index is -1, the box represents no class. This case demonstrates a situation where we want to evaluate the similarity between two class-groups by adding a class to one of the matched groups, but not both. Referring to Figure 12, the shaded boxes depict the order of the process. As shown, the algorithm proceeds as follows. It starts with adding C_{-1}^1 into the class-group CG_0^1 then it adds the class C_6^2 into the class-group CG_0^2 , comparing the two class-groups and checking the similarity improvement. Then it remove the class C_6^2 from the class-group CG_0^2 and adds to it the class C_{11}^2 , evaluating its similarity against CG_0^1 , checking the similarity improvement and comparing it against the

similarity improvement achieved when adding C_6^2 and the best is maintained. The algorithm then remove the class C_{11}^2 from the class-group CG_0^2 and adds to it the class C_{12}^2 , evaluating its similarity again against CG_0^1 , checking the similarity improvement and comparing it against the best similarity improvement achieved so far and the best is maintained. The class is added to the class-group for which it achieves the best similarity improvement.

6.5 Summary

In this chapter we presented a staged matching framework consisting of three stages. The focus of the first stage is one-to-one matching, where each class in the smaller model is matched to a distinct class in the other model with which it is most similar. The focus of the second stage is one-to-many matching, where, classes not passing the matching threshold of the first stage are tried to be combined in class-groups and a feasibility of the match from a single class in one model to a group of classes in the other model is investigated. The third stage is an extension of the second stage, which is meant to capture many to many matching. Empirical investigation for the matching framework is presented in Chapter 8.

CHAPTER 7

MODEL CONSOLIDATION

7.1 Introduction

Model merging is the task of unifying information in the input models together while keeping a single copy of matched elements [33]. Within the context of our framework we state the task of our merging operator as follows. Given, as input, a set of analysis (design) instances along with their pair-wise similarity information, the aim of our proposed merging algorithm is to generate, as output, an analysis (design) reference model with the following properties: 1) it represents all the input instances (completeness); 2) it must retain the granularity of the elements of the input instances; 3) each element in the reference model is traceable to its original instance (traceability); 4) each input instance can be instantiated back from the reference model (instantiation-ability); 5) it offers the reuse potential of the instances it generalizes; 6) it can give some guidance to the analyst about the best domain practices.

This chapter is organized as follows. In Section 7.2 we present some basic concepts and definitions. The phased merging is introduced in Section 7.3 and detailed in Sections 7.4 and 7.5. Section 7.6 discusses the reference model's properties.

7.2 Basic Concepts and Definitions

As mentioned in Chapter 4, for each pair of input models the three-staged matching algorithms produce, as output, the *MSM* matrix (Section 4.4.4) which maintains the matching similarity information between the matched classes of the two models of the pair, depicted as three similarity levels.

Definition 7.1: Let M_i and M_j , $0 \leq i < j \leq n$, be a pair of models whose elements are matched by the 3-stage matching algorithms. The Matching Similarity Matrix ($MSM_{i,j}$) represents the similarity information of the matched elements of the pair M_i and M_j at three level of similarities, *highly similar* (S), *similar with variation* (V), and *unmatched* (U), as identified by the 3-stage matching algorithms.

Table 13 shows the MSM matrices of each pair (M_i and M_j) of the four models M_0, M_1, M_2 , and M_3 of Figure 1.

Definition 7.2: Let $P_{1,2}^k$ be a pair of models M_1 and M_2 ; let c_i^1 and c_j^2 be two classes, where $c_i^1 \in M_1$ and $c_j^2 \in M_2$; the matched classes c_i^1 and c_j^2 have the similarity level S (*highly similar*) if their similarity score satisfies the similarity threshold defined in the first matching stage.

Example: referring to Table 13-(a), classes c_0^0 and c_0^1 are highly similar classes, given that the similarity threshold defined for the first matching stage is 0.8; similarly the classes c_3^0 and c_3^1 .

Definition 7.3: Let $P_{1,2}^k$ be a pair of models M_1 and M_2 ; let CG_I^1 and CG_J^2 be two class-groups, where $CG_I^1 \in M_1$ and $CG_J^2 \in M_2$; the matched class-groups CG_I^1 and CG_J^2 have the similarity level V (*similar with variation*) if their similarity score satisfies the similarity threshold defined in the second matching stage, but it did not satisfy the similarity threshold defined in the first matching stage.

Example: referring to Table 13-(b), class-group $CG_{\{1,5\}}^0$, which consists of two classes (C_1^0 and C_5^0) has similarity level S with matched class-group $CG_{\{1\}}^2$ which consists of just one class (C_1^2).

Definition 7.4: If two classes $C_i^1 \in M_1$ and $C_j^2 \in M_2$ have similarity level S in the MSM matrix, they represent *instances* of the same conceptual class C.

Example: referring to Table 13-(a), classes C_3^0 and C_3^1 are instances of the same conceptual class Reservation, see Figure 1.

Definition 7.5: If two class-groups $CG_I^1 \in M_1$ and $CG_J^2 \in M_2$ have similarity level “V” in the MSM matrix, they represent different variants of the same conceptual class C.

Example: referring to Table 13-(b), class-group $CG_{\{1,5\}}^0$ has similarity level V with the class-group $CG_{\{1\}}^2$, which are both instances of the conceptual class Flight, see Figure 1.

The commonalities and the variabilities between the models of each pair are identified based on the levels of similarity identified between the matched classes, where classes mutually identified across all the MSMs with similarity level S are modeled as common, elements mutually identified across all MSMs with similarity level S and/or V are modeled as variants, and elements with similarity level U are modeled as optional.

Definition 7.6: Let $C_0^0, C_1^1, \dots, C_n^n$ be the instances of a conceptual class C in models M_0, M_1, \dots, M_n ; these instances are modeled as a *common* class C in the reference model if they mutually have the similarity level S in all pairs of matched models.

Table 13. Pair-wise MSM Matrices of Models M0, M1, M2, and M3

M_0 classes	C_3^0	C_6^0	C_2^0	C_0^0	C_4^0	C_5^0	C_1^0	-
M_1 classes	C_3^1	C_7^1	C_2^1	C_0^1	C_4^1	C_5^1	C_1^1	C_6^1
Sim. Score	0.92	1	0.91	0.94	0.84	0.90	0.87	
Sim. level	S	S	S	S	S	S	S	U

(a) MSM matrix of pair M_0 & M_1

M_0 classes	C_3^0	C_6^0	C_2^0	C_0^0	C_4^0	$C_{\{1,5\}}^0$	-	
M_2 classes	C_3^2	C_6^2	C_2^2	C_0^2	C_4^2	C_1^2	C_5^2	C_7^2
Sim. Score	0.99	0.99	0.86	0.81	0.85	0.76	-	
Sim. level	S	S	S	S	S	V	U	U

(b) MSM matrix of pair M_0 & M_2

M_0 classes	C_3^0	C_6^0	C_2^0	C_0^0	C_4^0	$C_{\{1,5\}}^0$	
M_3 classes	C_3^3	C_5^3	C_2^3	C_0^3	C_4^3	C_1^3	
Sim. Score	0.94	0.99	0.85	0.96	0.85	0.78	
Sim. level	S	S	S	S	S	V	

(c) MSM matrix of pair M_0 & M_3

M_1 classes	C_6^1	C_7^1	C_2^1	C_3^1	C_4^1	C_0^1	$C_{\{1,5\}}^1$	-
M_2 classes	C_5^2	C_6^2	C_2^2	C_3^2	C_4^2	C_0^2	C_1^2	C_7^2
Sim. Score	1	0.99	0.94	0.91	0.82	0.83	0.75	
Sim. level	S	S	S	S	S	S	V	U

(d) MSM matrix of pair M_1 & M_2

M_1 classes	C_7^1	C_4^1	C_3^1	C_0^1	C_2^1	$C_{\{1,5\}}^1$	C_6^1	
M_3 classes	C_5^3	C_4^3	C_3^3	C_0^3	C_2^3	C_1^3	-	
Sim. Score	0.99	0.84	0.87	0.90	0.89	0.85		
Sim. level	S	S	S	S	S	V	U	

(e) MSM matrix of pair M_1 & M_3

M_2 classes	C_6^2	C_3^2	C_4^2	C_0^2	C_1^2	C_2^2	C_5^2	C_7^2
M_3 classes	C_5^3	C_3^3	C_4^3	C_0^3	C_1^3	C_2^3	-	-
Sim. Score	1.0	0.94	0.90	0.86	0.81	0.95		
Sim. level	S	S	S	S	S	S	U	U

(f) MSM matrix of pair M_2 & M_3

Example: referring to Table 13-(a-f), classes C_0^0 , C_0^1 , C_0^2 , and C_0^3 are kind of classes modeled as a common class in the reference model, as they are mutually highly similar (S) in all pairs of matched models.

Definition 7.7: Let the class-groups $CG_0^0, CG_1^1, \dots, CG_n^n$ be the instances of a conceptual class C in each of the models M_0, M_1, \dots, M_n , respectively; then, these instances are generalized as *variants* in the reference model, with a variation point, if they consistently and mutually have the similarity level **S** or **V** in all pairs of matched models, with at least one pair having the similarity level **V**.

Example: referring to Table 13-(b), class-group $CG_{\{1,5\}}^0 \in M_0$ is similar (**V**) to the class-group $CG_{\{1\}}^2 \in M_2$, therefore, they represent two different variants in the reference model, under the same variation point. On the other hand, $C_1^2 \in M_2$ has “S” similarity level with the class $C_1^3 \in M_3$, indicating that they are common within the corresponding pair and thus they will be modeled as the same variant in the reference model.

The basic underlying process for our proposed merging algorithm can be described, as follows. Common elements in the reference model are those elements mutually have “S” similarity level across all the pairs and they are represented by a single class in the reference model. Variants are modeled through Variation Points (VP) which act as interfaces for their different variants. Identical (highly similar) variants under the variation point are unified. Optional elements are modeled through Optional Points (OP) which act as interfaces for the different optional elements. Identical (highly similar) optionals under the optional point are unified. Each input model has a variant in each variation point, but it is not necessarily for each optional point to have an optional element for each input model.

7.3 Phased Merging

As mentioned in Section 4.4.6, merging is performed in two phases. Each phase is implemented in a staged manner. The focus of the first phase is to perform preliminary merging at the class level, producing a reference model preliminary catalog (RMPC) in which all the common, variant, and optional classes are identified across all the instances. The RMPC acts as a foundation for the second phase in which the union merge is performed at the level of attributes, methods, and relationships. The output of the second stage is the reference model catalog (RMC), from which the reference model, exemplified in Figure 2, is produced. Detailed description of the merging algorithms in both phases will be the focus of next two sections.

7.4 First Phase Merging

The first merging phase is preceded by a preprocessing mechanism through which some models will be filtered out, as not candidate for merge, while the rest are passed through to be consolidated in the reference model. Given n input models, candidate for merge, the first phase merging algorithm works on $\frac{n(n-1)}{2}$ MSM matrices, representing the matched elements similarity information in pair-wise manner. The merging starts by selecting one pair of models (say M_i and M_j) from those models candidate for merge and then merging them to create an initial reference model. Then the other models are merged to the reference model one at a time.

7.4.1 First Pair Selection

The first pair of models to be merged can be selected in many different ways. They can be selected randomly, based on the model sequence number as given by the tool when reading XMI file (model with smallest number is selected first), or based on their

similarity/dissimilarity to the other models as depicted in Figure 15. The merging algorithm should be deterministic, apart from the selection method adopted. However, as described in Section 7.4.2, the algorithm when performing merging has only a local view about the ultimate similarity level of the element in the reference model. In other words, an element could be found as common among the first set of merged models, but when the algorithm proceeds in merging the other models it may find that this element has the similarity level “V” (requiring modeling it as variant) or “U” (requiring modeling it as optional) in one of the MSM matrices. This will result in reconstructing the reference model during merging, which requires extra processing time. Similarly, an element could be found as variant among the first set of merged models, but when the algorithm proceeds in merging the other models it finds that this element has the similarity level “U” in one of the MSM matrices, requiring modeling it as optional in the reference model. This lack of the global view can be even worse with the elements to be modeled as optionals. Let us assume that a class C_1^k of model M_k is represented as two classes (C_1^l and C_2^l) in another model M_l and has no similar class(es) in a third model M_m . Let us assume that the algorithm selects M_l and M_m to be generalized first. The algorithm will find the similarity level of both C_1^l and C_2^l as “U”, for they exist in M_l but not in M_m . Thus, the two classes will be modeled as optional under two different optional points. Later on, when the algorithm merges M_k , it realizes that the similarity information between the classes of M_l and M_k indicates that the two classes C_1^l and C_2^l , combined, are representing a variant of the class C_1^k . This means that the two classes C_1^l and C_2^l need to be modeled as one optional variant under a single optional point instead of modeling

them as two optionals under two different optional points as happening when merging M_i and M_m . Thus, the two optional points needs to be merged into one optional point.

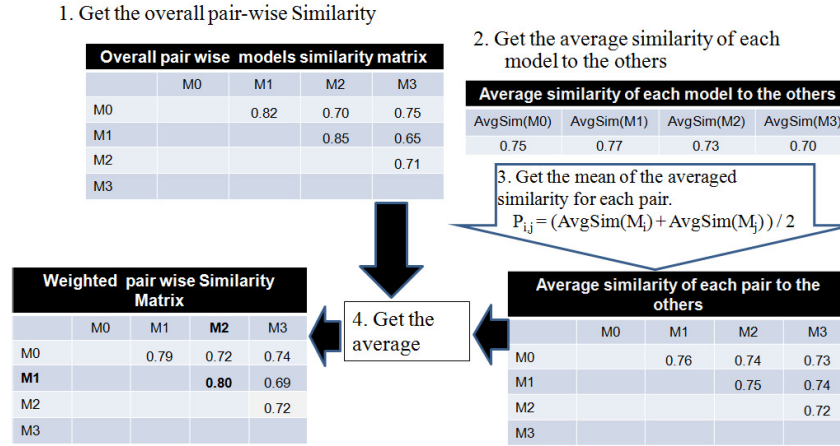


Figure 15. Selection of the First Pair for Merging

Alternatively, if the first pair of models to be generalized is selected based on the following.

- Minimizing the number of elements of similarity level “S”. Thus, we minimize changing common to variant or to optional during merging, as any common element must have a similarity level of “S” in any pair of models.
- Minimizing the number of elements of similarity level “U”. Thus we minimize merging more than one optional point into one as the case demonstrated above.
- Maximizing the number of elements of similarity level “V”. This criteria has twofold advantage: it minimizes changing common into variant. It also helps giving a better view about ultimate representation of the optional elements in the reference model (maximizing common will not help for this advantage).

Following this last approach the selection score (SS) for each pair of models (M_i and M_j) can be calculated from the following formula:

$$SS(M_i, M_j) = \frac{1}{N(S)} + \frac{N(V)}{N(V)+N(S)} + \frac{1}{N_i(U)+N_j(U)}, \quad (16)$$

where $N(S)$ is the number of matched elements between M_i and M_j with similarity level “S”, $N(V)$ is the number of matched elements between M_i and M_j with similarity level “V”, $N_i(U)$ is the number of unmatched elements (similarity level “U”) in model M_i , and $N_j(U)$ is the number of unmatched elements in model M_j .

7.4.2 Merging Algorithm

When the first pair of models is selected, the corresponding *MSM* matrix is retrieved and traversed by the consolidation algorithm as a consolidation guide to model the commonalities and variabilities in the reference model. Any classes with similarity level “S” in such a matrix are modeled as common classes in the reference model. A variation point is created in the reference model for each pair of matched class-groups with “V” similarity level; and each class-group in the pair is added as a different variant under that variation point. As per the classes that are marked as “U” in *MSM* matrix, they are temporally ignored if they are more than one class for certain model, to be considered later on and modeled as optional points. The reason for ignoring them can be reputed to the lack of information about these classes in the other models. For example, if two classes of certain model are found to be unmatched in the *MSM* matrix when merging the corresponding model(s), we do not know enough information about these classes, as they may represent two different conceptual classes or they may just represent one conceptual class. In the former case they need to be modeled under two different optional points in the reference model while in the later case they are modeled as just one option under one optional point. This information will not be clear until we get clear view about them from the other models. This case of an ambiguity happens mainly when the unmatched classes

for certain model are more than one. However, the case should be clear when the model has just one unmatched class. In this case an optional point is created. Then under this optional point the unmatched class is added as an optional class for the corresponding model.

After the first two models are merged to create an initial version of the reference model, the other models are merged to the reference model one at a time. The next model can be selected randomly or based on its similarity to the those models already merged in the reference model.

To make the idea of creating an initial version of the reference model clear we demonstrate it through a simple example. Assume that M_0 and M_1 are selected as the first pair to be merged. In Figure 16, the first column shows how our proposed algorithm merges the first two selected models, M_0 and M_1 , whose *MSM* matrix is given in Table 13-(a). As shown in Figure 16, the merge of M_0 and M_1 results in: *seven* classes modeled as *common*; *no variants*; and *one unmatched* class modeled as first optional point. The common classes are represented in the common matrix, which is depicted at the first column of Figure 16 with its rows represent the models and its columns represent the matched classes. Classes in the same column mutually have *S* similarity level in all the models already merged, and thus will be represented by just one single class in the reference model^{*}.

^{*} Each model instance that is merged in the reference model must have a class in each column of the common matrix and it must have *S* similarity level with all the classes in the column.

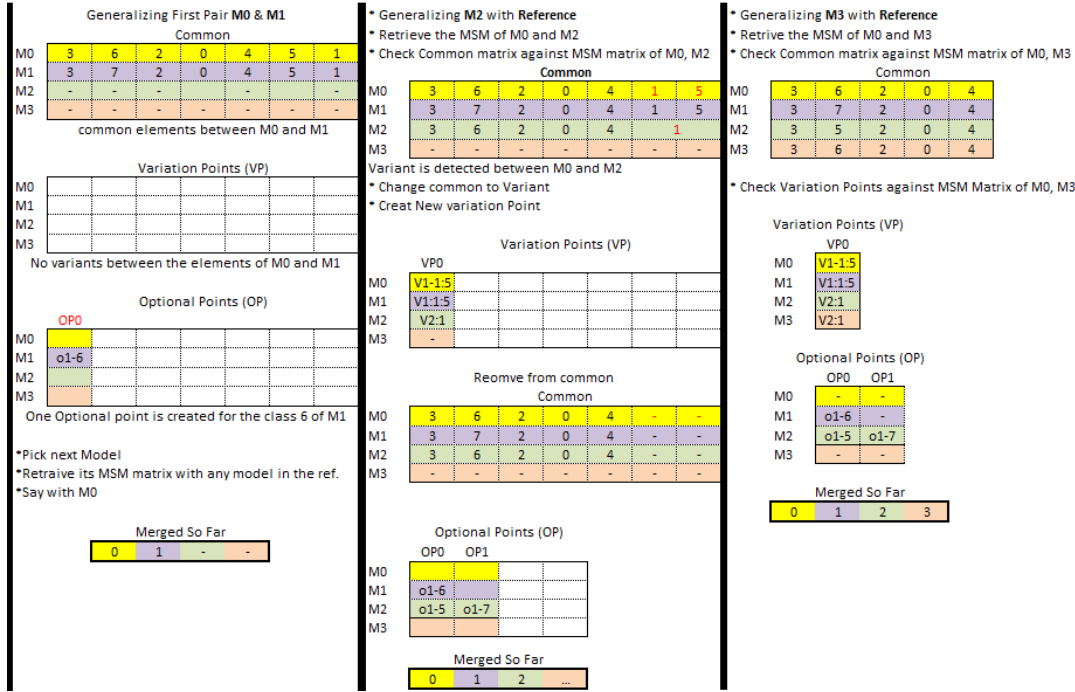


Figure 16. First Phase Merging Steps

For the unmatched class c_6^1 , a new optional point (OP_0) is created and the class is modeled as an optional class for model M_1 —represented as o_{1-6} , with “1” indicates the index of the optional and “6” indicates the index of the optional class in the original instance (that is in M_1)*.

After the first pair is merged to create an initial version of the reference model, the next model (say M_2 in our example) is selected and merged with the reference model as follows. First, the *MSM* matrix, representing the matched elements between the selected model and one of the models already in the reference model (say M_0), is retrieved and its similarity information is checked against the similarity information in the reference model. In particular, the similarity matrix, $MSM_{0,2}$, depicted in Table 13-(b), which represents the similarity of the matched elements between M_0 and M_2 , is retrieved and its

* Any optional point can optionally have classes from any instance model.

similarity information is compared against the similarity information of the reference model, which is depicted in the first column of Figure 16. The existence of M_0 information in both $MSM_{0,2}$ and the reference model similarity information acts as a tracer or a facilitator between the two pieces of information and thus resulting in a smooth way for modeling the commonality and variability when the new model is to be merged with the reference model. The role played by such a linkage can be summarized as follows. The Common matrix maintains the original indices of the common elements for each model merged so far to the reference model. The indices of the common elements corresponding to the linking model (M_0)* are traced in the $MSM_{0,2}$ matrix. The aim is to ensure that each element of M_0 , which is modeled as common in the reference model, has a matched element in M_2 and the two matched elements are identified as highly similar in $MSM_{0,2}$. If this is satisfied for each common element, nothing is done except that the indices of M_2 classes corresponding to the common elements are copied to the row of M_2 in the Common matrix. If, however, a common element of M_0 is identified as not highly similar to its matched element of M_2 , then action will be taken appropriately as will be detailed in the following. Referring to Table 13 and Figure 16, the classes c_0^0 , c_1^0 , c_2^0 , c_3^0 , c_4^0 , c_5^0 and c_6^0 of M_0 are modeled as common in the reference model. However, when tracing these classes in $MSM_{0,2}$, the classes $\{c_1^0, c_5^0\}$ are found to be matched, as a class-group, to the class c_1^2 with the similarity level “V”. This indicates that the class-group $CG_{\{1,5\}}^0$ and the class C_1^2 needs to be represented as variants in the reference model. Therefore, a new variation point, VP_0 , is created in the reference model with two variants. The first variant represents $CG_{\{1,5\}}^0$ and $CG_{\{1,5\}}^1$ of M_0 and M_1 , respectively, while the

* Any model already merged in the reference could be a linking model.

second variant represents c_1^2 of M_2 . This means that classes represented by the first variants need not to be in the common matrix any more, thus the corresponding columns are removed from the common matrix. This situation represents a case where the reference model classes can change from one similarity level (common) to another (variants), during merging, as more instances are exposed to the algorithm. As per the classes c_7^2 and c_5^2 the algorithm will detect (by searching the optional points in the reference model) that c_5^2 has “S” similarity level with the class c_6^1 (of model M_1), which is modeled as optional under the optional point $OP0$. Therefore the algorithm will model c_5^2 as the same optional variant under the optional point $OP0$. However, the class c_7^2 has neither “S” similarity level nor “V” similarity level with any class in the reference model. Therefore, a new optional point ($OP1$) is created and the class c_7^2 is modeled as an optional class under $OP1$.

The algorithm then proceed to generalize the next model (M_3) in the same manner as the case with M_2 . After retrieving the MSM matrix corresponding to M_0 and M_3 , the algorithm will start by cross-checking the common elements between the reference model and the new model with the help of the linking model (M_0) and the matching similarity matrix $MSM_{0,3}$ of the new model (M_3) and the linking model (M_0). Then the algorithm cross-checks the variants between the reference model and the new model. During the generalization of M_3 , the algorithm will find that the classes c_1^3 and c_5^3 of M_3 have, as one class-group, “V” similarity level with the class c_1^0 of M_0 . The algorithm will start searching for the class c_1^0 of M_0 in the common classes of M_0 in the reference to check whether c_1^0 is a common class in the reference or not. If it finds that c_1^0 is a common class, a new variation point is created and all the corresponding classes of the other

models, in the same column of c_1^0 , are modeled (along with c_1^0) as one variant under the new variation point while the corresponding matching class/class-group of the new model is modeled as another variant. In the case where the c_1^0 is not common in the reference the variation points are searched. If c_1^0 exists under any variation point VP_p then the variants under that variation point are searched with the hope of finding highly similar variant to the class/class-group of the new model. If this variant exists the new class/class-group is linked with such a variant. If, however, no such a variant exists, the new class/class-group is modeled as a new variant under the variation point VP_p . In our case the class c_1^0 exists under the variation point VP_0 and it has “S” similarity level to the class c_1^1 of M_1 , which is modeled as the second variant, v2-1, under VP_0 . Therefore the class c_1^0 is modeled as a second variant under the variation point VP_0 . No optional classes exists for M_3 .

Searching the variation or the optional points is very efficient as it will just search the entries corresponding to the linking model in the variation point matrix, which has a linear complexity time, i.e. $O(|VP|)$, in the worst case.

Table 14. Reference Model Preliminary Catalog (RMPC)

	Common					VPs	OPs	
						VP_0	OP_0	OP_1
M_0	C_3	C_6	C_4	C_2	C_0	V2- $\{C_1:C_5\}$	-	-
M_1	C_3	C_7	C_4	C_2	C_0	V2- $\{C_1:C_5\}$	o1-C_6	-
M_2	C_3	C_6	C_4	C_2	C_0	V1- C_1	o1-C_5	o1-C_7
M_3	C_3	C_5	C_4	C_2	C_0	V1- C_1	-	
RM	C_0	C_1	C_2	C_3	C_4	-	-	-

Table 14, represents the Reference Model Preliminary Catalog (RMPC), which summarizes the common, variant, and optional classes of our hypothetical example (Figure 1).

In summary, as the output of the matching algorithms are the *MSM* matrices, which identify what is common and what is variant between each pair of the input models, the output of the first merging phase algorithm is the RMPC, which generalizes the matching similarity information in all the *MSM* matrices.

7.5 Second Phase Merging

In the first merging phase, all the instances of the same class across the input instances are generalized into a single class in the reference model if they mutually have the similarity level **S**. However, since the matching is performed based on a threshold similarity, highly similar classes does not mean that they are identical. Some differences may exist at the attribute, operation, or relationship level. Since our goal is to maintain the granularity of variability and commonality at the finer grained granularity, we propose a second phase merging algorithm to handle such a generalization. Based on the RMPC, the actual catalog of the reference is built as follows. For each column j in the common part of RMPC a reference class C_j^r is created. Then all the attributes and the methods of the corresponding class of model M_0 are copied to C_j^r . Next, the corresponding classes of the other models sharing the same column are generalized one at a time, using union merging. Attributes (or methods) that exist in the classes of some instances but not in the others are tagged with a binary vector (called instance tag) in which the presence of 1 in the i^{th} location of the vector indicates the existence of the attribute in the corresponding class of the instance i , 0 indicates otherwise. Variant and

optional classes are generalized in the same way. Similarly, the relationships between the classes are generalized using union merging. The reference model built from this catalog is depicted in Figure 2.

7.6 Reference Model Properties

In Chapter 1, we listed a set of properties that characterize our proposed reference model. In the following we discuss how the reference model achieves these properties.

7.6.1 Reference Model Reuse

The reuse potential of the reference, as compared to the reuse from a single instance, is discussed in Section 8.8 (Experiment 6) .

7.6.2 Reference Model Completeness

As indicated in Chapter 1, reference model completeness means that if an element appears in one of the source models, it must be represented in the reference model as well. This simply means that information in the source models must not be compromised during merging. Our staged merging algorithms perform merging at different level of granularity. At the class level, i.e. first merging phase, common classes are unified while variants are explicated through variation or optional points. Variation points allow us to maintain the different alternatives so that they are not compromised. Optional points allow us to maintain those classes that exist in some models but not in the others. Doing so, our proposed representation of reference model preserves the design differences among the different input instances. The second phase merging algorithm allows us to maintain all the necessary information at finer grained of granularity. For example, although a class in the reference model has a single name, while representing many instances, the names of the classes in the different instances are maintained in the

reference class as aliases. Also the information at the level of attributes, methods, or relationships are maintained in the reference. Each common attribute across the different instances is maintained as a single attribute with no tag, indicating that it is present in every instance. Attribute that is common to some instances but not to others or specific to a certain instance is tagged with the instance tag which signifies the instances it represents. Same thing can be said about methods and relationships.

7.6.3 Reference Model Traceability and Instantiate-ability

The representation of the proposed reference model allows each instance to be instantiated back from the reference model. Common classes are part of every instance. Common (non-tagged) attributes or methods are part of every instantiated class. Referring to Figure 2, a relationship with a variant tag prefixed with “cc” means that it is between two common classes. Some relationships are prefixed with “cc”, but they are not part of every instance. Hence, instance tag indicates which instance a relationship represents. Variation points represents an abstraction between the different variants and the other classes in the model. Variation points and the optional points are not part of the instantiated instance. They are removed and the relationships connected directly with the corresponding variants with the help of both the instance tag and the variant tag. For example, in the reference model presented in Figure 2, the class “Plane” is connected to the variation point VP0 with an association relationship named “assigned to”, with an instance tag <1:1:1:1> and variant tag “cv: c1-VP0.v1/2.c8/10:0”. The instance tag indicates that this relation presents in all instances. The prefix “cv” in the variant tag indicates that the relationship is between a common class and a variant class. The common class is “c1” and the variant class is under the variation point VP0. It can be

“c8”, as variant “v1”, or it can be “c10” as variant “v2”. Let us assume we want to instantiate instance 2 — instances are numbered from 0 to $n-1$. Then, looking at the variant classes under the variation point VP0, we can see that from the instance tag labeling the relationships connecting the variation points with its variants, the class “Flight” is the corresponding class. Tracing the corresponding variant tag x^* -VP0.v1.c8: x^* , we find it matches to one alternative “~~cv:c1-VP0.v1/2.c8/10:0~~” in the variant tag “cv: c1-VP0.v1/2.c8/10:0”. Therefore, the class “Plane” has an “assigned to” association relationship with the class “Flight” in instance 2.

7.6.4 Reference Model Reuse Recommendations

The instance vector annotating the reference model elements, while helping in tracing the elements back to their original instances, can serve as an indicator for the commonality of each element across the individual instances generalized by the reference. This commonality will guide the reuser about the common analysis and design practices in the domain.

7.7 Summary

In this chapter we presented a phased merging framework consisting of two phases. The focus of the first phase is to perform preliminary merging at the class level, producing the reference model preliminary catalog in which all the common, variant, and optional classes are identified across all the instances. The focus of the second phase is to perform merging at the finer level of granularity, i.e. at the level of attributes, methods, and relationships, producing the reference model catalog. Empirical investigation for the merging algorithms is presented in Chapter 8.

CHAPTER 8

EMPIRICAL ANALYSIS

8.1 Introduction

In this chapter we empirically validate the proposed staged consolidation framework. The chapter is organized as follows. Section 8.2 introduces the experimental objects used in our empirical investigation along with our empirical investigation road map. In Section 8.3 we present the matching accuracy measures used. The proof of concept tool is presented in Section 8.4. The weight calibration experiments are discussed in Section 8.5. In Section 8.6, we compare the performance of the proposed greedy GA matching algorithm (GGAM) against the traditional GA. We validate the comparison framework along with the matching algorithms in Section 8.7. Empirical investigation of the reference model generalization is presented in Section 8.8.

8.2 Experimental Objects

The experimental objects for our empirical investigation need to be constrained to the objective of our work, *the generalization of a set of models, representing different instances within a domain or similar domains, into a reference model that unifies their overlaps and explicates their differences*. Therefore, the criterion of the instances suitable for our experiments is that models need to be realistic enough to manifest the best practices in both the industry and the academia, so that the theoretical reuse potential can be obtained and consequently the potential of our approach will be realized. Finding large mature data set available for research at the model level is difficult. Finding multiple mature model instances representing different variants of an application within a domain is exceedingly difficult. Therefore, the potential of our approach will be shown through a

couple of experiments, each is targeting certain facet. The following case studies are going to be used as our experimental objects.

Case Study 0 (CS0): This case study represents different variations of a simple flight booking system adopted from [44]. The variations were inspired from the different design alternatives introduced by the author while explaining the UML practice in modeling the structural view of the software system. It is used throughout the thesis as a hypothetical example to demonstrate interaction of the different components of the solution framework.

Case Study 1 (CS1): This case study represents within a domain class diagrams reversed engineered from an open source system, ezmorph^{*}, consisting of 12 releases. To allow for differences between the reversed engineered class diagrams of the different releases, we picked 5 non-consecutive releases (0.8, 0.9, 1.0, 1.0.4, and 1.0.6) of this system.

Case Study 2 (CS2): This case study, borrowed from [144], represents across domain class diagrams consisting of four class diagrams with similar structures (as they represent the admission systems) but in different ontologies (Computer Repair Shop, Hospital registration, Student Admission, and Admission in a General Institution). The structural similarity between the diagrams is very high, representing the reuse potential that should be reflected in the proposed reference model.

Case Study 3 (CS3): This case study consists of multiple instances instantiated by introducing different types of perturbation to an original model. The original model is borrowed from Case Study 1, consisting of 50 classes. The perturbations by the instance

^{*} <http://sourceforge.net/projects/ezmorph/?source=directory>

generator are applied at different level of granularity (classes, relationships, attributes, methods, and data types). Table 15 shows the different types of perturbations applied to the original model to generate the different instances.

Table 15. Perturbation Performed by the Instance Generator

Class level perturbation		<i>pp</i>	<i>pc</i>
<i>renameClass</i>	Changing class name by perturbing their names with some prefix or suffix added from a predefined set of names.	0.50	NA
<i>removeClass</i>	Removing a class from the original model.	0.80	NA
Attributes perturbation			
<i>pertAttributesList</i>	Adding an attribute to a class from a predefined sets of attributes along with their data type. Changing the data type of the attribute	1.0	25%-30%
<i>removeAttributes</i>	Removing an attribute from a class	1.0	25%-30%
Operations list perturbation			
<i>pertOperationsList</i>	Adding an operations to a class from a predefined sets of operations along with their returns types and parameters. Adding parameter to the operation Changing the return type of the operation	1.0	25%-30%
<i>removeOperation</i>	Removing an operation from a class	1.0	25%-30%
Relationships perturbation			
<i>pertRelationship</i>	Adding relationships between two classes. Changing the relationship type between two classes. Changing the relationship's name between two classes.	0.15	NA
<i>removeRelation</i>	Removing a relationship from the original model.	0.15	NA

To allow for differences between the instances the perturbation operations are applied probabilistically. Two parameters are used by the generator, perturbation probability (*pp*) and percentage of changes (*pc*). The *pp* parameter represents the probability by which certain type of perturbation will be applied whereas the *pc* parameter represents the magnitude of such perturbation. For example, for the type of change *removeAttributes*, setting the *pp* parameter into 0.50 and the *pc* into 20% means that attributes will be

removed from the class, by the instance generator, with a probability of 0.50, and the number of removed attributes is 20% of the number of attributes of the class. It is worth mentioning that when applying some perturbation, undesirable situations may happen. For example, *removeClass* perturbation may result in splitting the class diagram into fragments. Similar thing can happen when removing some relationships. For such situations preventive actions are taken to not perform that perturbations. In other words, if removing the class and its relationships will result in splitting the class diagram into two or more fragments, that class will not be removed. Figure 17 shows a trace matrix of the classes' distribution over the different instances generated. The first line shows the class index in the original (source) model. Shaded boxes represent the classes which exist in the source model, but removed, by *removeClass* perturbation operation, from the corresponding instance.

Figure 17. Trace Matrix Showing Classes' Distribution over Different Instances, Case Study 3.

Table 16. Basic Statistics about the Case Studies

	Number of class diagrams	Number of pairs	Number of classes in the largest model	Number of classes in the smallest model
Case Study 0	4	6	8	6
Case Study 1	5	10	71	49
Case Study 2	4	6	10	10
Case Study 3	5	10	32	29

Table 16 provides basic statistics about the four case studies. Table 17 summarizes our empirical investigation road map. It shows, for each experiment, the dataset used, and the objective of the experiment.

Table 17. Empirical Investigation Roadmap

Experiment	Objective	Dataset used
Experiment 1	Setting the neighborhood weights within and across domains.	One pair (two models), selected from Case Study 2.
Experiment 2	Setting the name, internal, and neighborhood weights for equations 12 through 15. Showing the limitation of the single measure through 0 weight assignment for the other 2 measures.	One pair (two models) selected from Case Study 1. One pair (two models) selected from Case Study 2. One pair (two models) selected from Case Study 3.
Experiment 3	Evaluating the performance of the traditional genetic algorithm versus the performance of the greedy genetic algorithm.	Synthetic data. Case Study 1.
Experiment 4	Evaluating the accuracy of the different similarity metrics against the three matching algorithms presented in the first matching stage: GGRM; GGAM; and GSAM.	Case Study 1 (within domain). Case Study 2 (across domain). Case Study 3 (within domain).
Experiment 5	To show that the unrelated models will be filtered out and the reference will be built based on the majority of the instances . Evaluating the merging algorithms.	Mixing 2 instances from Case Study 0 with 4 instances from Case Study 2. Case Study 0 (Hypothetical Example). Case Study 2 (across domain).
Experiment 6	Evaluating the merging algorithms. Evaluating the reference reuse.	Case Study 1 (within domain). Case Study 3 (random instances).

8.3 Accuracy Measures

From the matching prospective, an accurate similarity assessment should result in an accurate matching, i.e., a matching with zero false positive and zero false negative rates. In other words, elements correctly matched by the matching algorithm should have been assigned high similarity scores by the similarity metric, so that they can pass matched/unmatched threshold, to be counted as true positives. However, elements incorrectly matched by the algorithm, due to injectivity (Definition 5.2), should have been assigned low similarity scores by the similarity metric so that they can be counted as true negatives, because of their low similarity values.

The accuracy of the similarity metrics and the matching algorithms are evaluated in terms of the matching *precision*, *recall*, and *accuracy*. It is a general problem that evaluating the accuracy of the matching depends heavily on the particular matching goal [54, 145, 146]. Within the context of the goal of this work, we will consider all pairs more similar than certain threshold to be matched, and all pairs less similar to be not matched [147]. Therefore we can define the three measures as follows. Let TP be the number of true positives (i.e. number of pairs of classes, correctly matched, with similarity score above or equal to the matching threshold), TN be the number of true negatives (i.e. the number of classes in each model that are correctly unmatched, or are matched, incorrectly, due to injectivity, but with low similarity score), FP be the number of false positives (i.e. number of pairs of classes incorrectly matched with similarity score above or equal to the matching threshold), FN be the number of false negative (i.e. the number of pairs of classes incorrectly unmatched, or matched correctly with low similarity score), then:

$$Precision(\%) = 100 \times \frac{TP}{TP + FP} \quad (17)$$

$$Recall(\%) = 100 \times \frac{TP}{TP + FN} \quad (18)$$

$$Accuracy(\%) = 100 \times \frac{TP + TN}{TP + TN + FP + FN} \quad (19)$$

8.4 Proof of Concept Tool

We developed a proof of concept java-based tool to implement the different algorithms presented in our framework and to show the applicability of our proposed solution and its potential. The tool receives as input a set of class diagrams in XMI (XML Metadata Interchange) format. The tool can then perform the following tasks.

- ✓ Parssing the XMI files as produced by two modeling tools: Altova and ArgoUML.
- ✓ Computing different similarity metrics with configurable weight settings. Currently the tool supports the metrics presented in Chapter 5. Other new metrics can be defined, coded, added, and called as seprate functions. The input to the similarity metric function is the information of a pair of compared classes or models, i.e. two versions are implemented for each similarity function. The output is either a single real value, represinting degree of similarity of a pair of classes, or a matrix of real values, represinting the pair-wise degree of similarity between the classes of the compared pair of models. To interface with the WordNet database we adopted, with some modefication, an open source package, ws4j [148].
- ✓ Matching the elements of the input models in a pair-wise manner. For element to element matching, the tool provides an implementation for five model matching algorithms: Simple Greedy Matching algorithm (SGRM), Globl Greedy (GGRM) Matching algorithm, traditional Genetic Matching algorithm (GA), Greedy Genetic Matching algorithm (GGAM), and Greedy Simulated Annealing Matching

algorithm (GSAM). For the second and third stages the tool implements the algorithms described in Section 6.3 & 6.4.

- ✓ Consolidating the models to build the reference model. Filtering is a preprocessing step performed by the tool to filter out unrelated models. The filtering is performed by the tool as described in Section 4.4.5 and shown experimentally in Section 8.8 (Experiment 5). Then, the tool in the first phase of merging produces the the Reference Model Preliminary Catalog (RMPC), which identifies the commonality and variability across the merged models at the class level, as described in Section 7.4. Then it goes for the second phase of merging, i.e. the merge of methods, attributes, and relationships.

8.5 Empirical Weights Investigation

In this experiment, we investigate different weight assignments for the constituents of the different compound metrics used for assessing the similarity between the elements of the compared models. We run different types of experiments for setting the values of the weight coefficients of the constituents of the compound metrics, Equations 11 through 15.

Experiment 1: Setting neighborhood similarity weights

Objective: To select the most appropriate weights for the different constituents (metrics) of the similarity metric *NSim*, Equation (11), so that each class in a certain model will be matched to the most similar class in the other model, based on the *NHS* similarity metric.

Methodology: The experiment was conducted according to the pseudo code in Figure 18. A pair of models is randomly selected from Case Study 2. Certain matching threshold is

defined. For each weight assignment, the similarity score for each pair of classes from the two models is computed, and the injective match from each class in the smaller model to its most similar, unmatched, class in the other model is found. The resulting match is evaluated in terms of the matching accuracy, Equation (19).

```

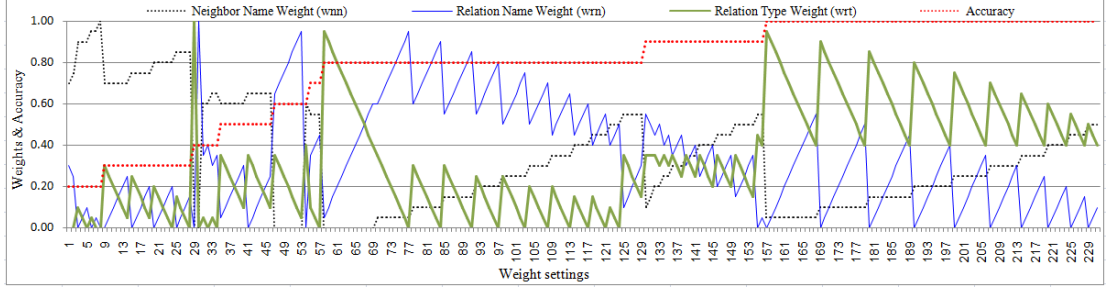
Pick a pair of models  $M_i$  and  $M_j$ 
for  $w_{nn} \leftarrow 0.0$  to  $1.0$  step  $0.05$ 
  for  $w_{rn} \leftarrow 0.0$  to  $1.0 - w_{nn}$  step  $0.05$  {
     $w_{rt} \leftarrow 1.0 - (w_{nn} + w_{rn})$ ;
    find  $NHS$  between the classes of models  $M_i$  and  $M_j$  based on
     $w_{nn}, w_{rn}, w_{rt}$  weights and store the similarity scores in  $ES$ 
    matrix;
    evaluate the matching accuracy between the classes of the
    models  $M_i$  and  $M_j$ ;
  }
end for
end for

```

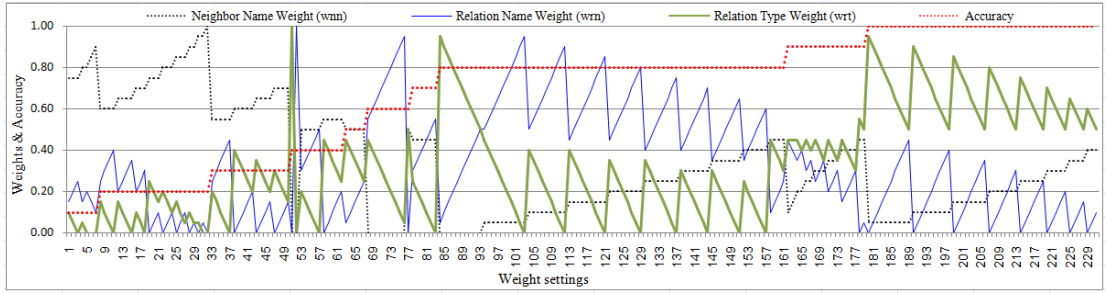
Figure 18. Pseudo Code of the Weight Calibration of the Constituents of $NSim$ Metric, Equation (11).

Figure 19 shows the obtained matching accuracy for different weight settings at different matching thresholds (0.70, 0.75, 0.80). The different experiments of the weight settings label the x -axis while y -axis values represent the values of the weight coefficients (w_{nn} , w_m , w_{rt}) along with the matching accuracy. We use the decimal point style for the accuracy, rather than percentage (%), to be in the same scale of the weights. We use the *Microsoft Excel Line Chart* type, which allows us to draw the trends of the accuracy versus the weight values over the different experiments of weight settings. The data series (weight coefficients & accuracy) in the diagrams are sorted increasingly by the accuracy. As we can see, the general trend in the three figures, Figure 19-(a) through Figure 19-(c), is that high accuracy was obtained when the weights assigned to both the neighbor name (w_{nn}) and the relation name (w_m) are low as compared to higher weight values assigned to

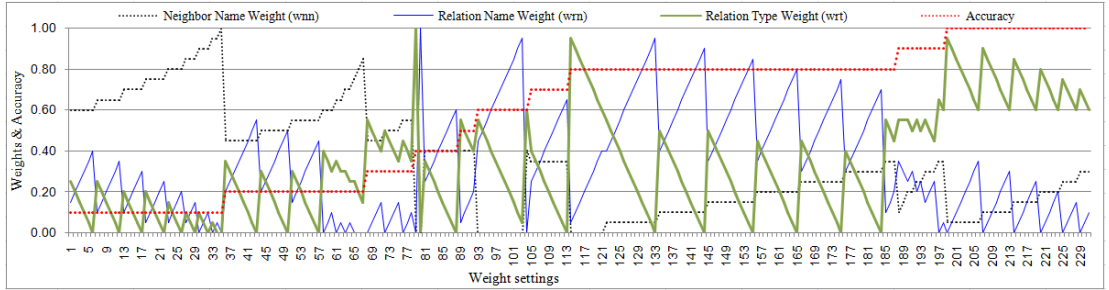
the relation type (w_{rt}). The reverse is also true, as the low accuracy was obtained when w_{rt} has low weights as compared to higher weights assignment to w_{nm} .



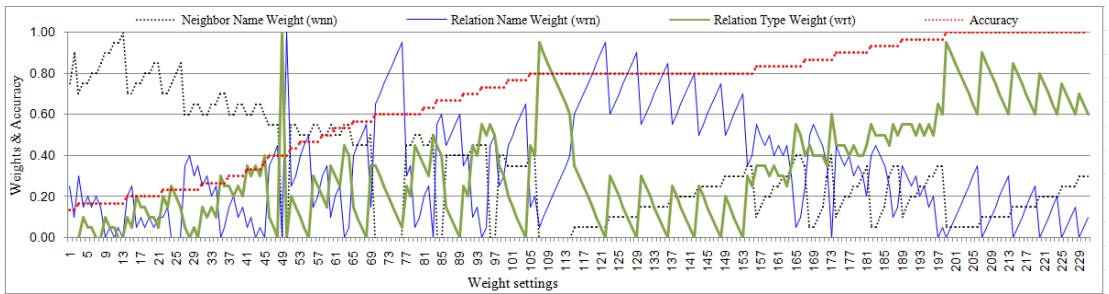
(a) 70% matching threshold



(b) 75% matching threshold



(c) 80% matching threshold



(d) Average of a,b, and c

Figure 19. Models' Matching Accuracy at Different Weight Settings for the Neighborhoods Similarity Metric NHS ' Constituents, Equation (11).

Table 18. The Accuracy Obtained at Some Special Cases of the Weight Assignment, *Nsim* Metric, Equation (11)

w_{nm}	w_{rn}	w_{rt}	Matching threshold				comments
			0.70	0.75	0.80	.85	
0	0	1	0.40	0.40	0.40	0.40	This reflects the importance of the other contributors, as using the relation type alone achieves only 40% accuracy over the different thresholds.
0	1	0	0.40	0.40	0.40	0.40	This reflects the importance of the other contributors, as using the relation name alone achieves only 40% accuracy over the different thresholds.
1	0	0	0.20	0.20	0.10	0	Relying on the neighbor name alone, across ontologies, resulted in 100% loss in the accuracy at higher threshold.
0	0.50	0.5	0.80	0.80	0.60	0.60	The absence of the neighbor name caused a loss of 40% in the accuracy, at higher threshold .
0.33	0.33	0.33	0.90	0.90	0.90	0.20	Even distribution of the weights resulted in a very poor accuracy at higher threshold.
0.50	0	0.50	1	0.60	0.30	0.20	The absence of the relation name caused a loss of 80% in the accuracy at higher threshold.
0.50	0.50	0	0.80	0.40	0.20	0.10	The absence of the relation type caused a loss of 90% in the accuracy at higher threshold
0	0.05-0.60	0.40-0.95	0.80	-	-	-	Best accuracy obtained and the corresponding weight ranges when using only relation name and relation type. As we can see here that as threshold goes up (from 0.7 up to 0.85) the range of weights which gives us high accuracy is getting smaller.
0	0.05-0.50	0.50-0.95	-	0.80	-	-	
0	0.10-0.40	0.60-0.90	-	-	0.80	-	
0	0.05-0.25	0.75-0.95	-	-	-	0.80	
0.05-0.60	0	0.40-0.95	1	-	-	-	Best accuracy obtained and the corresponding weight ranges when using only neighbor name and relation type. The absence of the relation name does not affect the accuracy.
0.05-0.50	0	0.50-0.95	-	1	-	-	
0.05-0.30	0	0.70-0.95	-	-	1	-	
0.05-0.25	0	0.75-0.95	-	-	-	1	
0.05-0.50	0.50-0.95	0	0.80	-	-	-	Best accuracy obtained and the corresponding weight ranges when using only neighbor name and relation name.
0.05-0.40	0.60-0.95	0	-	0.80	-	-	
0.05-0.30	0.70-0.95	0	-	-	0.80	-	
0.05-0.25	0.75-0.95	0	-	-	-	0.80	
0.05-0.50	0.05-0.55	0.40-0.90	1	-	-	-	Best accuracy obtained and the corresponding weight ranges when all constituents have nonzero weights.
0.05-0.40	0.05-0.45	0.50-0.90	-	1	-	-	
0.05-0.30	0.05-0.35	0.60-0.90	-	-	1	-	
0.05-0.25	0.05-0.25	0.70-0.90	-	-	-	1	
Note:	All the weights assignment is subject to the condition that the summation of all the weights is 1.						

Figure 19 (d) shows the average accuracy over the different matching thresholds for the same point of weight settings. Table 18 shows some weights' values that reflect some special cases, like even distribution of the weights; the absence of one constituent; the situation where only one constituent is used; and the situation where the best accuracy was obtained. As it is clear from Figure 19, and summarized by Table 18, that when the three constituents of Equation (11) are assigned even weights we got a high accuracy of 0.90 at 0.70, 0.75, and 0.80 matching threshold, but when the matching threshold was increased from 0.80 to 0.85 the accuracy was drastically decreased into 0.2. This can be attributed to the fact that across domains the lexical similarity between the names of the matched classes is low, resulting in a similarity lower than the threshold (increasing the number of false negative), which in turn results in decreasing the accuracy. At the case of relying on a single component of *NSim*, the best accuracy of 40% was obtained with the relation type ($w_{rt}=1$). This means that 60% of the accuracy was lost because of the absence of the other components ($w_{nn}=0$, $w_{rn}=0$). This is a clear evidence about the importance of the other constituents of *NSim*. The situation is not that worst with the absence of one component as the best accuracy of 100% was obtained with the absence of the relation name. However, the absence of one of the other two, i.e. $w_{nn}=0$ or $w_{rn}=0$, results in a loss of 20% in the accuracy.

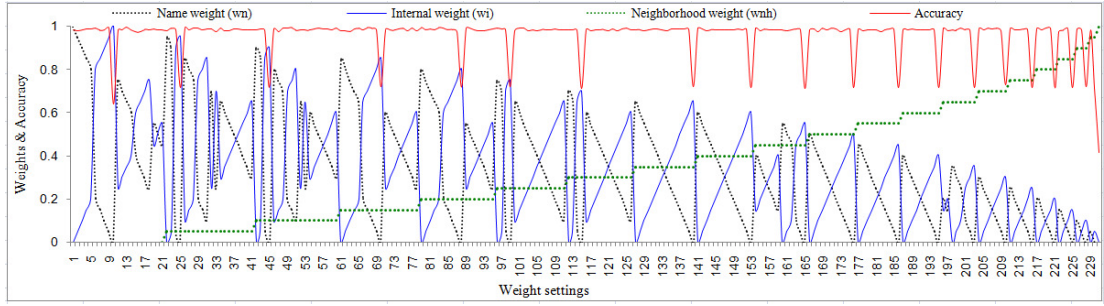
When all the three constituents are present (i.e. all have nonzero values for the weight coefficients), an accuracy of 100% can be obtained at ranges of weights shown at the end of Table 18. It is clear from Figure 19 and the last rows in Table 18 that we can still get a 100% accuracy at higher matching thresholds. However, the range of the weight assignments for the three constituents is getting smaller as the matching threshold is

getting higher. Bearing in mind the expected variations among the similar elements, the very high threshold may be so restrictive, resulting in an increase in the false negatives. In other words, having a high matching threshold can cause some similar classes, with some variation, to be identified as dissimilar, because the small variation between them render their similarity value to not pass the very high threshold. On the other hand, having a low matching threshold can result in high false positives. That is to say, having a low threshold can cause some dissimilar classes, with low similarity values, to be identified as similar. Therefore, we opt to adopt a reasonable threshold of 0.80 for our further experiments. For this threshold, the ranges of the weights which result in a 100% accuracy are: $w_{nn} \in \{0.05, 0.10, \dots, 0.30\}$; $w_{rn} \in \{0.05, 0.10, \dots, 0.35\}$; $w_{rt} \in \{0.65, 0.70, \dots, 0.90\}$. Taking the median within each set (conditioning that $w_{nn} + w_{rn} + w_{rt} = 1$) we can suggest the following weight settings: $w_{nn} = 0.15$; $w_{rn} = 0.15$; $w_{rt} = 0.70$.

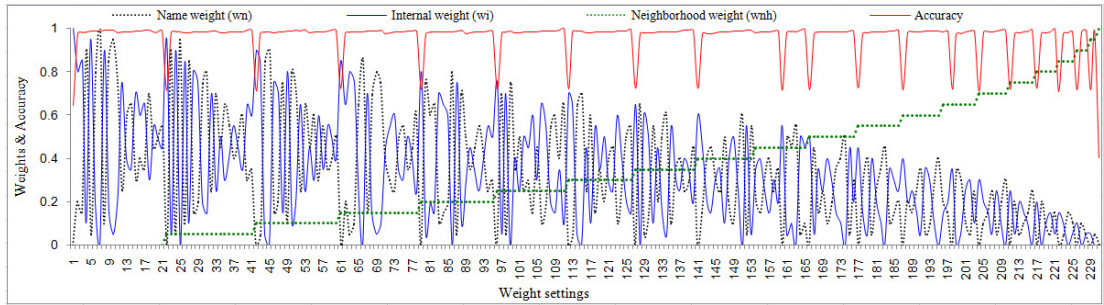
Experiment 2: Setting class similarity weights

Objective: To select the most appropriate weights for the different constituents of the similarity metrics $NIS, NNHS, INHS, NINHS$, Equation 12 through 15, so that each class in a certain model will be matched to the most similar class in the other model.

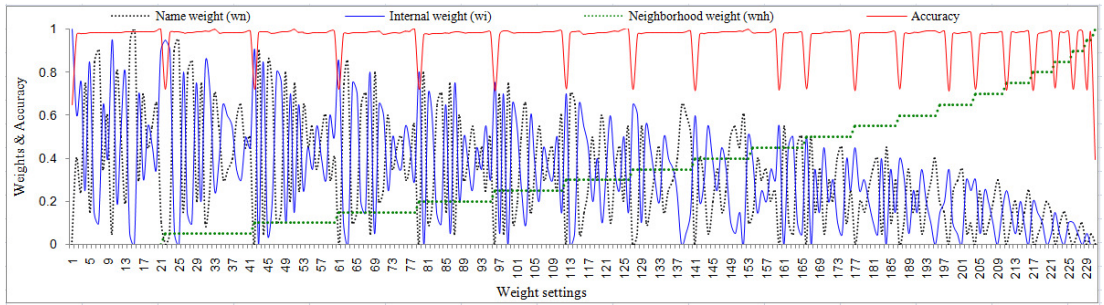
Experimental Objects: For this experiment we use Case studies 1, 2, and 3, to see how the weights will be calibrated over the different datasets.



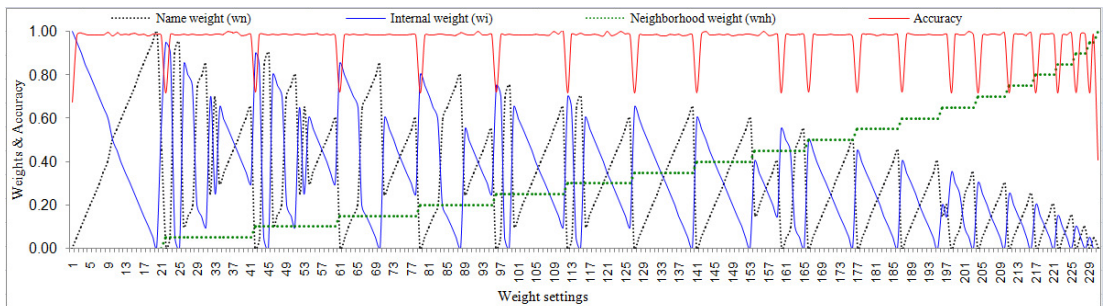
(a) 70% matching threshold



(b) 75% matching threshold



(c) 80% matching threshold



(d) Average of a,b, and c

Figure 20. Models' Matching Accuracy at Different Weight Settings for the *NINHS* Similarity Metric Constituents, Equation 12 through 15, Case Study 1

Methodology: The experiment was conducted as follows. For each case study, a pair of two models was randomly selected. For each threshold, the weights were assigned

according to the pseudo code in Figure 21. For each weight assignment, the similarity score for each pair of classes from the two models is computed, and the injective match from each class in the smaller model to its most similar, unmatched, class in the other model is found. The resulting match is evaluated in terms of the matching accuracy, Equation (19). It is easily to notice that Equations (12) through (14) are special cases from an Equation (15), where the weight coefficient for the missed constituent is zero. Therefore, the weight calibration experiments for the metric *NINHS* (Equation (15)) cover the weight settings for the four compound metrics, *NIS*, *NNHS*, *INHS*, and *NINHS*.

```

Pick a pair of models  $M_i$  and  $M_j$ 
for  $w_n \leftarrow 0.0$  to  $1.0$  step  $0.05$ 
  for  $w_i \leftarrow 0.0$  to  $1.0 - w_n$  step  $0.05$  {
     $w_{nh} \leftarrow 1.0 - (w_n + w_i)$  ;
    find NINHS between the classes of models  $M_i$  and  $M_j$  based on  $w_n, w_i, w_{nh}$ 
    weights and store the similarity scores in ES matrix;
    evaluate the matching accuracy between the classes of the models  $M_i$ 
    and  $M_j$ ;
  }
end for
end for

```

Figure 21. Pseudo Code of the Weight Calibration of the Constituents of *NINHS* Metric, Equation (15)

Figure 20 shows the obtained matching accuracy at different weight settings for the *NINHS* constituents for Case Study 1 (within domain class diagrams). The different experiments of the weight settings label the *x-axis* while *y-axis* values represent the values of the weight coefficients (w_n, w_i, w_{nh}) along with the matching accuracy. The data series (*y-axis* variables) in the charts are sorted by the values of the neighborhood weight coefficient, w_{nh} , increasingly. The reason for doing so is solely that it gives a clear view about the trend of the accuracy against each weight coefficient, as compared to sorting them by the accuracy, which is the case in Figure 19. As it is clear from the four figures

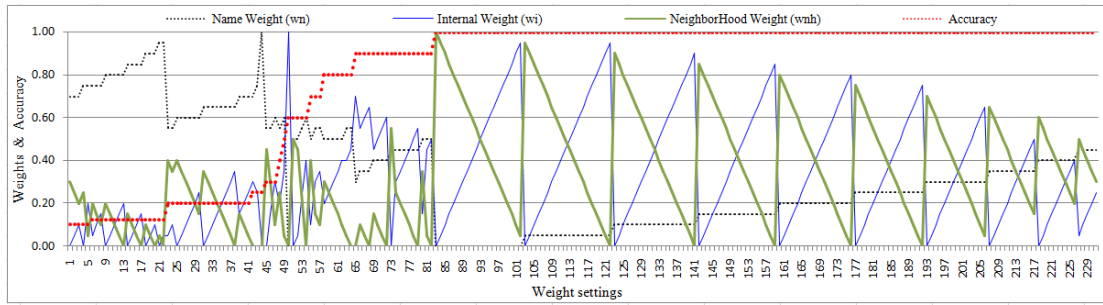
(Figure 20-(a) through Figure 20-(d)) that the drops in the accuracy happen when the weight assigned to the class name is 0, (i.e. when $w_n=0$). Since we opt to set the matching threshold to 0.80 in our further experiments of matching, and since the trend is the same at the different matching thresholds, Figure 20-(a) through Figure 20-(d), and for the sake of conciseness, our discussion will focus in Figure 20-(c) which is not far from other figures (i.e. Figure 20-(b) through Figure 20-(d)).

It is clear from Figure 20-(c) that when w_n is assigned any weight value ($w_n \geq 0.05$, such that $w_n + w_i + w_{nh}=1$) we usually get high matching accuracy between the classes of the matched class diagrams. The highest matching accuracy of 100% was obtained at different weight values, e.g. $\{w_n= 0.10; w_i= 0.50; w_{nh}=0.40\}$, $\{w_n= 0.75; w_i= 0.15; w_{nh}=0.10\}$ or $\{w_n= 0.55; w_i= 0.15; w_{nh}=0.30\}$. The worst accuracy of 39.4% was obtained at the weight settings $w_n= 0.0; w_i= 0.0; w_{nh}=1.0$. Table 19 summarizes some special cases of the weight assignment for the coefficients of Equations (12) through (15).

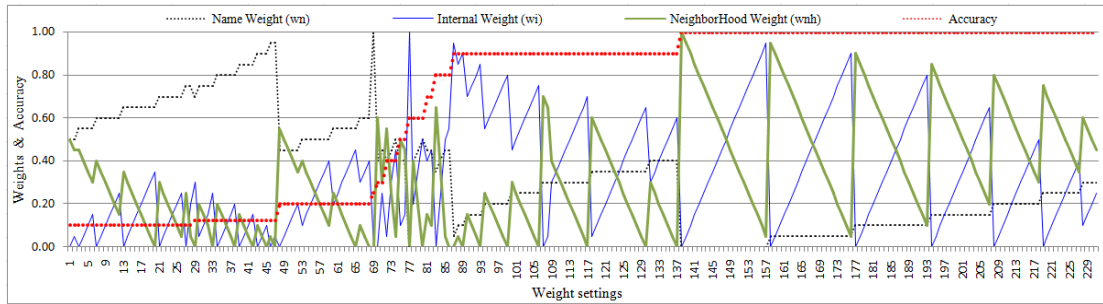
Figure 22 depicts the weights calibration and the corresponding accuracy across domains (Case Study 2) for equations (12) through (15). Special cases of these weight assignments are summarized in Table 20. As it is clear from the four figures (Figure 22 (a) through Figure 22 (d)) the drops in the accuracy happen when the weight assigned to the class name is high ($0.40 < w_n \leq 1.0$, such that $w_n + w_i + w_{nh}=1$). When w_n is assigned low weight values ($0 \leq w_n \leq 0.25$), we usually get high matching accuracy between the classes of the matched class diagrams. The highest matching accuracy of 100% was obtained at the weight values: $w_n= \{0.0, 0.05, 0.1, 0.15\}$; $w_i \in \{0.0, 0.05, \dots, 0.65\}$; and $w_{nh} \in \{0.55, 0.60, \dots, 1.0\}$; such that $w_n + w_i + w_{nh}=1$. The worst matching accuracy of 10% was obtained at the weight settings $w_n= 0.50; w_i= 0.0; w_{nh}=0.50$.

Table 19. The Accuracy Obtained at Some Special Cases of the Weight Assignment for the Metrics *NIS*, *NNHS*, *INHS*, *NINHS*, Equation (12) through (15), Case Study 1

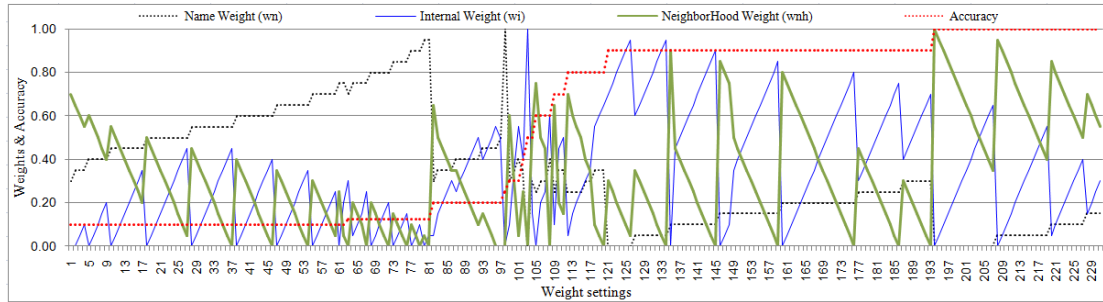
w_n	w_i	w_{nh}	Accuracy	comments
0	0	1	39.4%	This reflects the importance of the other contributors, as using the neighborhood information alone (<i>NHS</i> metric) achieves only 39.4% accuracy over the different thresholds.
0	1	0	65%	Relying on the internal information alone (<i>IS</i> metric) achieves only 65% accuracy over the different thresholds.
1	0	0	97%	Relying on the class name alone (<i>NS</i> metric), within domain, resulted in 97% accuracy. This can be understood as the lexical naming similarity between classes of multiple releases is expected to be high.
0	0.50	0.50	72%	The absence of the class name caused a loss of 28% in the accuracy.
0.50	0	0.50	97%	The absence of the internal information caused a loss of 3% in the accuracy.
0.50	0.50	0	99%	The absence of the neighborhood information with even weight assignment of the other weight coefficients caused just a loss of 1% in the accuracy.
0.33	0.33	0.33	98%	Even distribution of the weights for all the weight coefficients resulted in an accuracy of 98%.
0	0.15	0.85	72.8%	Best accuracy obtained and the corresponding weight ranges when $W_n = 0$. The absence of the class name caused a loss of 27.2% in the accuracy. This situation represent the best weight settings for <i>INHS</i> metric, Equation (14).
{0.1, 0.25, 0.4, 0.6, 0.75}	0	{0.1, 0.25, 0.4, 0.6, 0.75}	99%	Best accuracy obtained and the corresponding weights when $W_i = 0$. The absence of the internal information caused unnoticeable loss in the accuracy. This situation represent the best weight settings for <i>NNHS</i> metric, Equation (13).
0.10	0.90	0	99%	Best accuracy obtained and the corresponding weight ranges when $W_{nh} = 0$. The absence of the neighborhood information caused unnoticeable loss in the accuracy. This situation represent the best weight settings for <i>NIS</i> metric, Equation (12).
{0.10, 0.20, 0.25, 0.55, 0.75}	{0.15, 0.25, 0.30, 50}	{0.05, 0.10, 0.15, 0.30, 0.40, 0.45}	100%	Best accuracy obtained and the corresponding weights when all constituents have nonzero weights. This situation represents the best weight settings for <i>NINHS</i> metric, Equation (15).



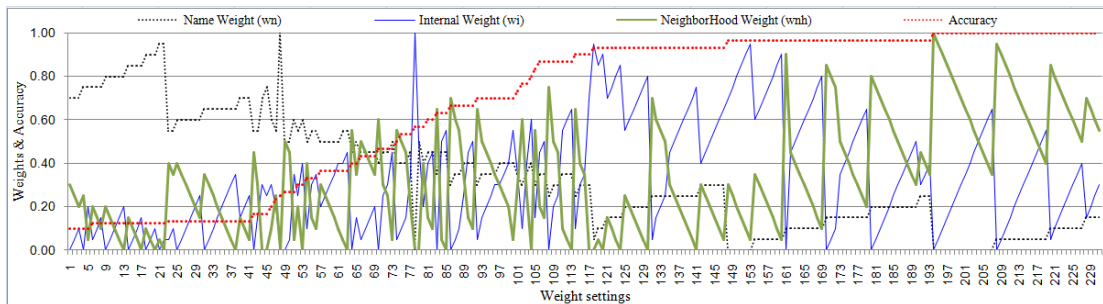
(a) 70% matching threshold



(b) 75% matching threshold



(c) 80% matching threshold



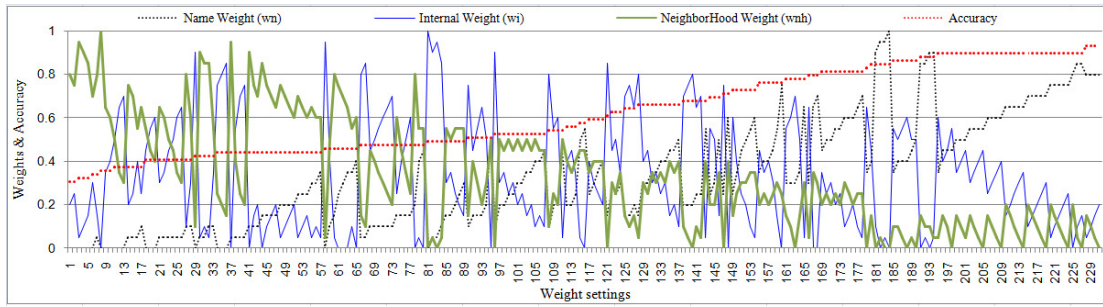
(d) Average of a, b, and c

Figure 22. Models' Matching Accuracy at Different Weight Settings for the *NINHS* Similarity Metric Constituents, Equation 12 through 15, Case Study 2

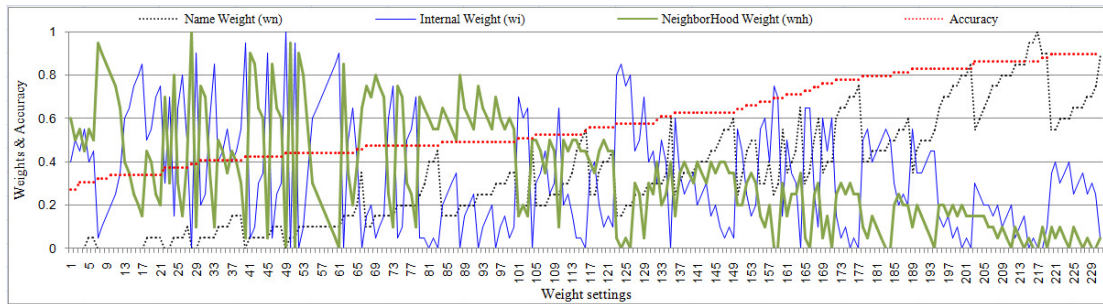
Table 20. The Accuracy Obtained at Some Special Cases of the Weight Assignment for the Metrics *NIS*, *NNHS*, *INHS*, *NINHS*, Equation (12) through (15), Case Study 2

w_n	w_i	w_{nh}	Accuracy	comments
0	0	1	100%	This reflects the importance of the neighborhood information across domains, as using the neighborhood information alone (<i>NHS</i> metric) achieves 100% accuracy.
0	1	0	50%	Relying on the internal information alone (<i>IS</i> metric) achieves only 50% accuracy.
1	0	0	25%	Relying on the class name alone (<i>NS</i> metric), across domains, reported an accuracy of 25%.
0	0.50	0.50	100%	The absence of the class name information with even weight assignment of the other weight coefficients does not cause any loss in the accuracy.
0.50	0	0.50	10%	The absence of the internal information with even weight assignment of the other weight coefficients caused a loss of 90% in the accuracy.
0.50	0.50	0	20%	The absence of the neighborhood information with even weight assignment of the other weight coefficients caused a loss of 80% in the accuracy.
0.33	0.33	0.33	40%	Even distribution of the weights for all the weight coefficients caused a loss of 80% in the accuracy.
0	0.05-0.65	0.35-0.95	100%	Best accuracy obtained and the corresponding weight ranges when $w_n = 0$. The absence of the class name result in no accuracy loss. This situation represents the best weight settings for <i>INHS</i> metric, Equation (14).
0.05	0	0.95	100%	Best accuracy obtained and the corresponding weights when $w_i = 0$. The absence of the internal information caused no accuracy loss. This situation represent the best weight settings for <i>NNHS</i> metric, Equation (13).
0.05-0.30	0.70-0.95	0	100%	Best accuracy obtained and the corresponding weight ranges when $w_{nh} = 0$. The absence of the neighborhood information caused unnoticeable loss in the accuracy. This situation represent the best weight settings for Equation (14).
{0.05, 0.10, 0.15}	0.05-0.55	0.40-0.90	100%	Best accuracy obtained and the corresponding weights when all constituents have nonzero weights.

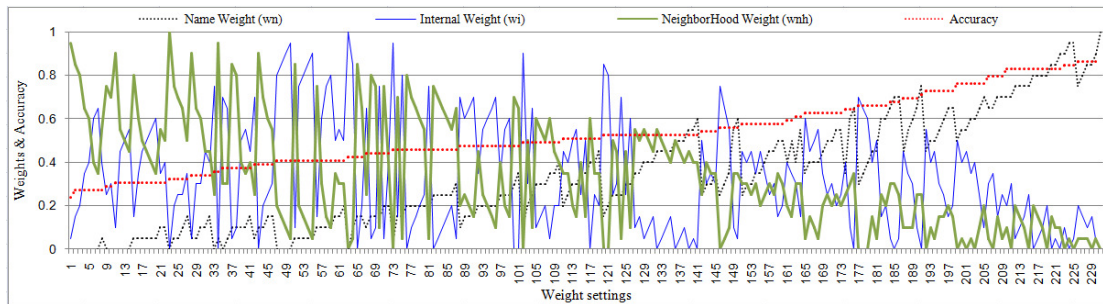
Figure 23 depicts the weights calibration and the corresponding accuracy based on Case Study 3, for equations (12) through (15). Special cases of these weight assignments are summarized in Table 21.



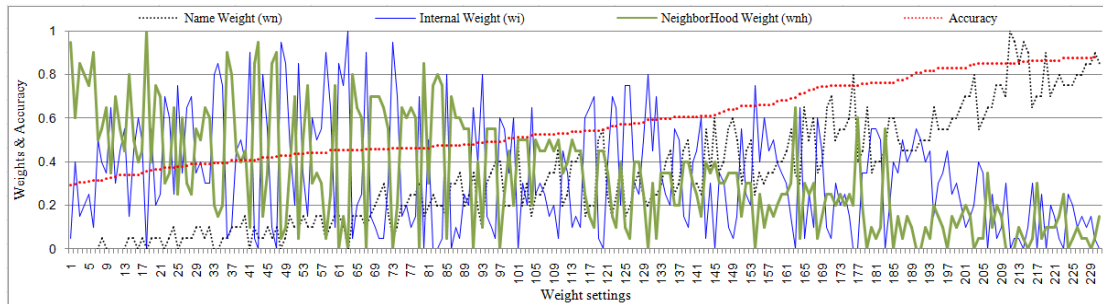
(a) 70% matching threshold



(b) 75% matching threshold



(c) 80% matching threshold



(d) Average of a, b, and c

Figure 23. Models' Matching Accuracy at Different Weight Settings for the *NINHS* Similarity Metric Constituents, Equation 12 through 15, Case Study 3

Table 21. The Accuracy Obtained at Some Special Cases of the Weight Assignment for the Metrics *NIS*, *NNHS*, *INHS*, *NINHS*, Equation (12) through (15), Case Study 3.

w_n	w_i	w_{nh}	Accuracy	comments
0	0	1	32.2%	This reflects the importance of the other contributors, as using the neighborhood information alone (metric <i>NHS</i>) achieves only 32.2% accuracy over the different thresholds.
0	1	0	42.4%	Relying on the internal information alone (metric <i>IS</i>) achieves only 42.4% accuracy over the different thresholds.
1	0	0	86.4%	Relying on the class name alone (metric <i>NHS</i>), within domain, resulted in 86.4% accuracy.
0	0.50	0.50	30.5	The absence of the class name information with even weight assignment of the other weight coefficients caused a loss of 69.5% in the accuracy.
0.50	0	0.50	52.5%	The absence of the internal information with even weight assignment of the other weight coefficients caused a loss of 47.5% in the accuracy.
0.50	0.50	0	76.3%	The absence of the neighborhood information with even weight assignment of the other weight coefficients caused just a loss of 23.7% in the accuracy.
0.33	0.33	0.33	55%	Even distribution of the weights for all the weight coefficients resulted in an accuracy of 55%.
0	0.80-0.95	0.05-0.20	40.7%	Best accuracy obtained and the corresponding weight ranges when $W_n = 0$. The absence of the class name caused a loss of 27% in the accuracy. This situation represent the best weight settings for <i>INHS</i> metric, Equation (14).
0.95	0	0.05	84.7%	Best accuracy obtained and the corresponding weights when $W_i = 0$. The absence of the internal information caused unnoticeable loss in the accuracy. This situation represent the best weight settings for, <i>NNHS</i> metric, Equation (13).
0.85	0.15	0	86.4%	Best accuracy obtained and the corresponding weight ranges when $W_{nh} = 0$. The absence of the neighborhood information caused unnoticeable loss in the accuracy. This situation represent the best weight settings for <i>NIS</i> metric, Equation (12).
0.75-0.90	0.05-0.20	0.05	86.4%	Best accuracy obtained and the corresponding weights when all constituents have nonzero weights. This situation represent the best weight settings for <i>NINHS</i> metric, Equation (15).

As it is clear from the four figures (Figure 23-(a) through Figure 23-(d)), the trend is generally similar to the situation with Case Study 1, Figure 20. High accuracy is achieved when w_n is assigned high values, against low values assigned to w_{nh} . On the other, When

w_n is assigned low values we get low accuracy. Table 21 provides concise comments about different weight assignments. As we can see, with Case study 3, and under the perturbation settings in Table 15, we did not obtain a 100% accuracy. The reason can be explained as follows. Assume a class A in model M_a was identical to a class B in the other model M_b (i.e. A and B are generated from the same class in the original model). Assume the perturbation has been applied to the two classes according to perturbation settings in Table 15. Assume a third class C in M_j generated from a different class in the original model. It is possible that, due to the high perturbation, the class A becomes more similar to C than it is to B , after perturbation. This will result in a miss, as our measure for reporting the accuracy is based on tracing the matched classes back to their original class to report whether the match is correct or not.

We modified the perturbation as shown in Table 22. Then, we rerun the experiment and the results at 0.80 threshold is shown in are shown in Figure 24. The best accuracy of 100% is achieved at the weight values: $w_n = \{0.65, 0.70, \dots, 0.90\}$; $w_i \in \{0.10, 0.15, \dots, 0.25\}$; and $w_{nh} \in \{0.0, 0.05, \dots, 0.15\}$; such that $w_n + w_i + w_{nh} = 1$.

Table 22. Low perturbation, Case Study 3

Perturbation type	pp	pc
<i>renameClass</i>	0.20	NA
<i>removeClass</i>	0.80	NA
<i>pertAttributesList</i>	0.50	20%-25%
<i>removeAttributes</i>	0.50	20%-25%
<i>pertOperationsList</i>	0.50	20%-25%
<i>removeOperation</i>	0.50	20%-25%
<i>pertRelationship</i>	0.10	NA
<i>removeRelation</i>	0.10	NA

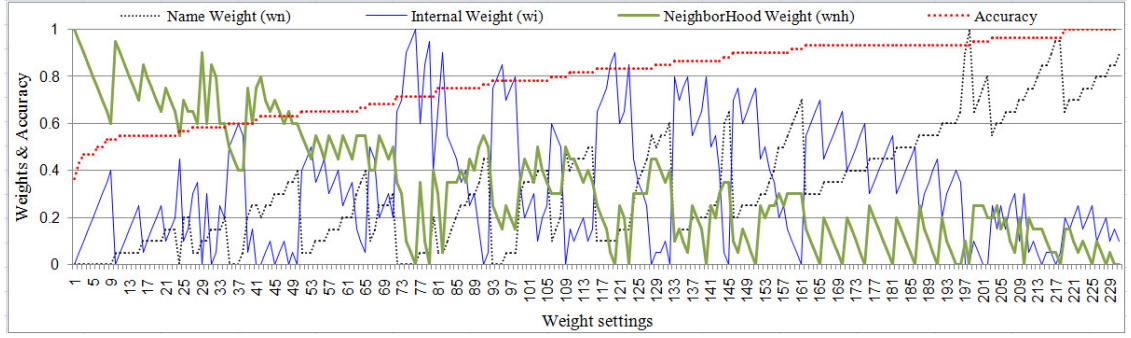


Figure 24. Models' Matching Accuracy at Different Weight Settings for the NINHS Similarity Metric Constituents, Equation 15, Case Study 3 (Low Perturbation)

To sum up, we can say that within a domain lexical information has more and recognized importance than the structural one. However, across domains structural information are more effective. The following section will validate our outcomes of experiment 1 and 2.

8.6 Empirical Investigation of Traditional Genetic versus Greedy Genetic

To compare the implementations of the traditional genetic (GA) and the hybridized greedy genetic algorithm (GGAM), we ran a couple of experiments as follows.

Experiment 3: Evaluating the performance of the traditional versus the greedy-genetic algorithms.

Experimental objective: To show the effect of the hybridization on the algorithm convergence.

Experimental objects: Due to the common problem of real data scarcity, and since we want to investigate the two algorithms under different problem sizes, the two algorithms are first investigated using synthetic data. Then they are investigated using Case Study 1. The synthetic data generator has been designed in such a way that it adheres to the theory of the problem domain [149] as well as our intuition about the problem. In other words,

in software engineering literature, the theoretical reuse potential within a domain can be up to 85% (65% as domain specific and 20% as domain independent) [34]. Accordingly, we devised an algorithm to generate data where the simulated similarity is within the theoretical potential. In other words, the generator generates a two dimensional matrix, which simulates the similarity scores between the elements of a pair of two models, where the randomly generated scores reflect the fact that an element of a certain model is dissimilar (has low similarity score) to all elements in the other model except one or two elements at most (high similarity score). Java code for the synthetic data generator is shown in Figure 25, and an example of the generated matrix, which simulate the element similarity matrix ES , is depicted in Figure 25, with problem size $n=10$.

Methodology: the experiments were run over different problem sizes ($n = 10, 20, 30, 40, 50$, and 100). The accuracy of the two algorithms is reported in term of the value of the fitness function and its closeness to the theoretical value. The fitness function is computed as the summation of the scores of the mapped elements, i.e., $\sum_{i=0}^{n-1} ES_{i,j}$, where i represents the index of the row element, j represents the index of the column element mapped to i ; $ES_{i,j}$ is the simulated similarity score between i and j ; and n simulates the number of classes in the two mapped models. We also reported the run time of the two algorithms at different problem sizes. The two algorithms ran under settings mentioned in Table 12, except the number of iterations which we set here, for the purpose of this experiments, to be 20,000 iterations, and for all the problem sizes.

Results and analysis: Figure 27 shows the convergence of the fitness function, to the theoretical value, for both the traditional genetic (right side of the figure) and the greedy genetic (left side of the figure) over different problem sizes. The theoretical value is the

sum of highest values in each row of the simulated *ES* matrix. In all the figures (Figure 27-(a) through Figure 27-(l)) the x-axis represents the different generations of the solution while y-axis represents the values of the fitness function.

```

1. static void generateRandSimMat (double a[], int row, int col)
2. {
3.     int randSim;
4.     double rand;
5.     for (int i=0; i<a.length; i++){
6.         // select random column element
7.         randSim=(int) (Math.random()*col);
8.
9.         for (int j=0; j<a[0].length; j++){
10.            // generate a random number to simulate the similarity score
11.            // between element i and element j
12.            a[i][j]=Math.random();
13.            // simulate dissimilarity by making the score smaller
14.            a[i][j]=a[i][j]*a[i][j];
15.
16.            // simulate high similarity by imposing high score for 75% of
17.            // row elements each with randomly selected column element
18.            if (j==randSim && a[i][j]<0.9 && Math.random()<=0.75){
19.                while((rand=Math.random())<0.9);
20.                a[i][j]=rand;
21.            }
22.        }
23.    }
24. }
25.

```

Figure 25. Synthetic Data Generator for ES Matrix

The following can be noticed from Figure 27-(a) through Figure 27-(l).

- 1- At low problem size ($n=10$), the two algorithms show almost equivalent convergence, yet, GGAM converges faster (in earlier iterations) than does the traditional GA, and with slightly higher value for the fitness function. However, none of the two algorithms reaches the theoretical value. The reason could be due to maintaining the injectivity. For example, referring to the simulated *ES* matrix in

Figure 26, the best solution is shown as bolded numbers, where the row elements are mapped to the corresponding column elements with best fitness value of $(0.952 + 0.974 + 0.911 + 0.900 + 0.920 + 0.982 + \mathbf{0.253} + \mathbf{0.214} + 0.984 + 0.852 = 7.92)$. The value 0.253, which simulates the similarity between the 7th row element ($i=6$) and the 3rd column element ($j=4$), is not the highest in its corresponding row, as the highest value is 0.778, but the algorithm enforcedly (to maintain injectivity) maps the 7th row element with the 3rd column element in favor of maximizing the overall fitness value. Similar thing can be said about the value 0.214. This difference between the highest values and the best option to go with by the algorithm can be the main cause for not getting to the theoretical fitness value. Other possible reason could be the fact that the algorithm could not converge to the optimal solution.

- 2- As the problem size is getting larger, the difference in the convergence between the two algorithms becomes clear, where GGAM converges faster (i.e., in earlier iterations) to the theoretical value while the traditional GA still needs more iterations to converge to the same value obtained by GGAM.
- 3- Time wise,

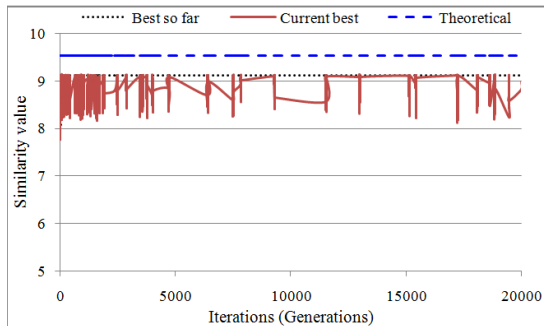
- 4- Table 23 shows the best values achieved by the traditional GA, up to 20000 iteration. The table also shows the corresponding time to achieve these values as taken by the two algorithms. It could be better to measure the time taken by the two algorithms to the best value achieved by either of the two algorithms, which is usually achieved by the GGAM across the different problem sizes. However, looking at Figure 25, we can see that the convergence of the traditional GA to the best values achieved by the GGAM cannot be figured out. In other words, we do not know how many iterations, beyond 20000 iterations, the traditional GA requires to converge to the best values achieved by the GGAM. As per the penalty that might be encountered due to the hybridization, in the case the algorithms will converge over the same numbers of iterations, Table 24 shows the time taken by the two algorithms over 20000 iterations, and across the different problem sizes.
- 5- The fluctuation in the range of the current solution, obtained by the two algorithms, over the different generations, at different problem sizes, shows the contribution of the greedy idea to limit the randomness involved in the traditional GA. Figure 28 shows the convergence behavior of the two algorithms over the first 200 iterations at a problem size of 50.

		j elements									
		0	1	2	3	4	5	6	7	8	9
i elements	0	0.222	0.241	0.130	0.097	0.121	0.211	0.671	0.267	0.952	0.759
	1	0.313	0.373	0.656	0.170	0.002	0.974	0.726	0.327	0.127	0.705
	2	0.626	0.570	0.650	0.375	0.801	0.344	0.204	0.911	0.741	0.720
	3	0.741	0.185	0.459	0.048	0.900	0.002	0.002	0.114	0.162	0.829
	4	0.302	0.920	0.000	0.303	0.334	0.001	0.000	0.039	0.782	0.281
	5	0.204	0.395	0.622	0.982	0.601	0.125	0.656	0.795	0.503	0.684
	6	0.778	0.608	0.253	0.026	0.442	0.393	0.081	0.140	0.081	0.000
	7	0.162	0.071	0.025	0.134	0.392	0.076	0.214	0.012	0.036	0.338
	8	0.606	0.435	0.593	0.064	0.673	0.032	0.527	0.805	0.747	0.984

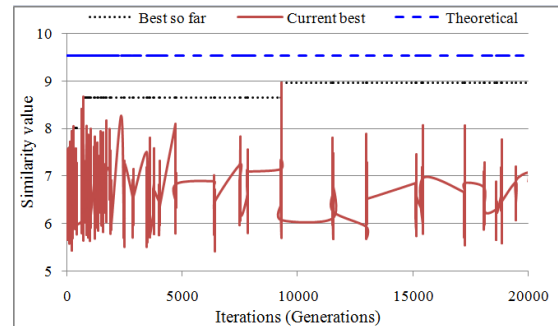
9	0.852	0.032	0.107	0.565	0.553	0.316	0.001	0.530	0.291	0.618
----------	--------------	-------	-------	-------	-------	-------	-------	-------	-------	-------

Figure 26. Example of the Simulated ES Matrix Generated by the Synthetic Data Generator

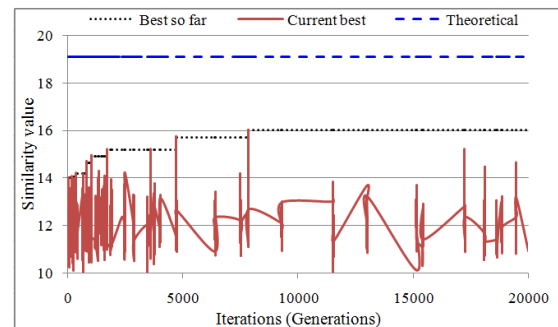
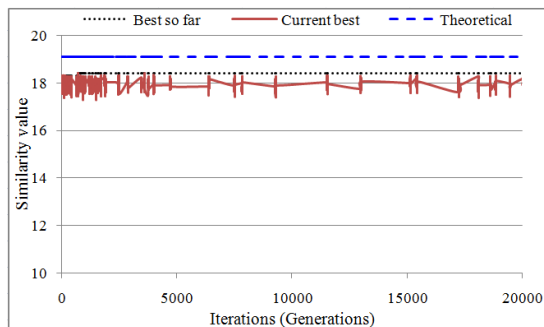
Figure 29 compares the performance of the traditional genetic against the greedy genetic in terms of the matching accuracy using real data, Case Study 1. The figure shows the results of two implementations of the traditional genetic. The first implementation shows the matching accuracy of the GA when the algorithm use only Roulette Wheel (RW) as a selection method. The second shows the matching accuracy of the GA when the algorithm maintain best 50% individuals to the next generation while selecting the other 50% using Roulette Wheel. The worst performance among the three algorithms was obtained when the Roulette Wheel method alone was used as a selection method. Maintaining the top 50% of the solutions while selecting the other 50% using the RW method improved the accuracy significantly. Again as it is the case in Figure 27 the GGAM is the superior among the three algorithms.



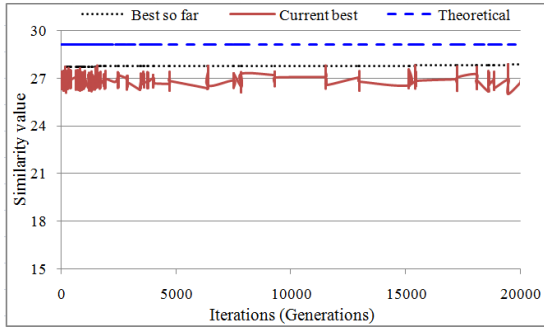
(a) Performance of GGAM, n=10



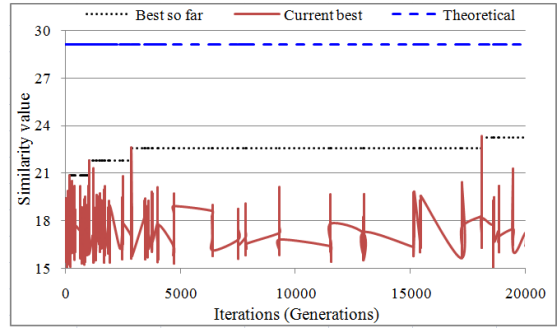
(b) Performance of traditional GA, n=10



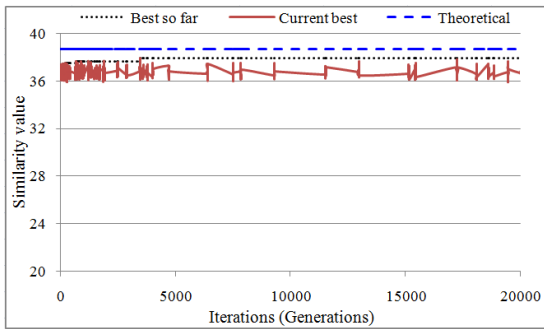
(c) Performance of GGAM, n=20



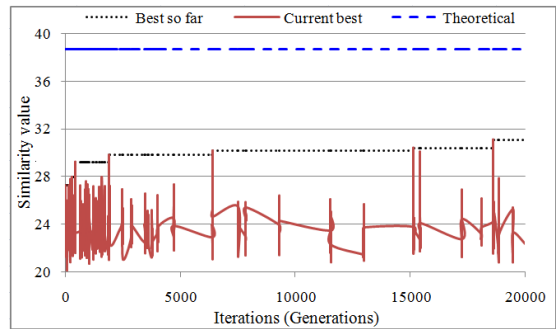
(d) Performance of traditional GA, n=20



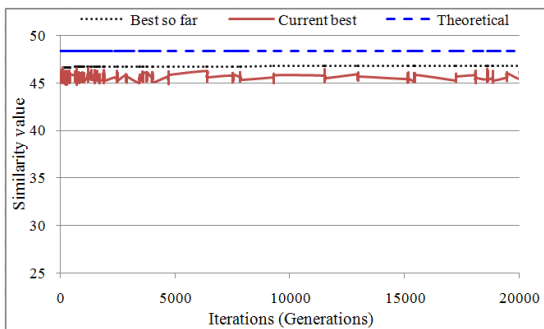
(e) Performance of GGAM, n=30



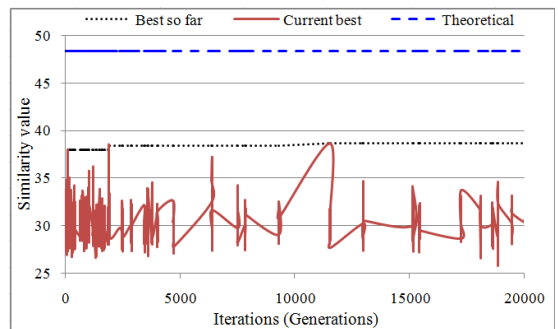
(f) Performance of traditional GA, n=30



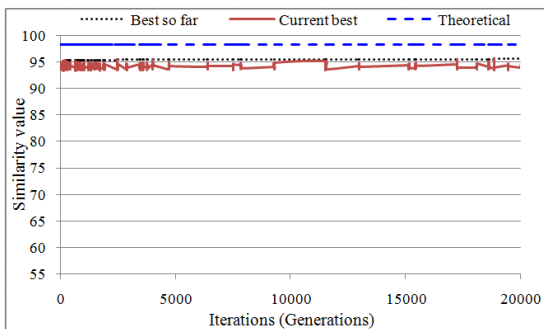
(g) Performance of GGAM, n=40



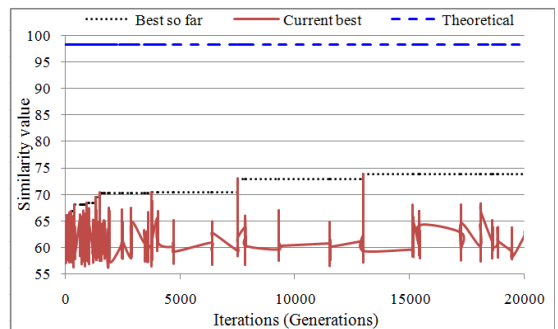
(h) Performance of traditional GA, n=40



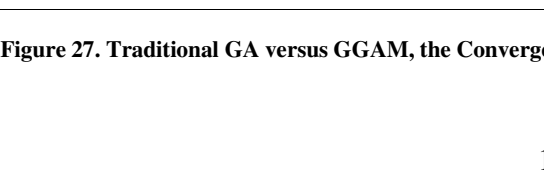
(i) Performance of GGAM, n=50



(j) Performance of traditional GA, n=50



(k) Performance of GGAM, n=100



(l) Performance of traditional GA, n=100

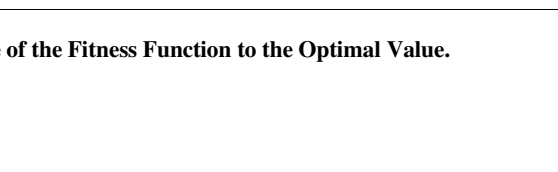


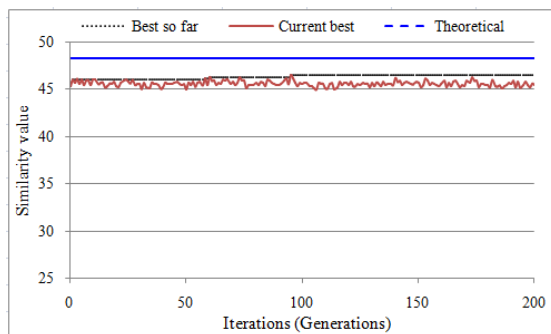
Figure 27. Traditional GA versus GGAM, the Convergence of the Fitness Function to the Optimal Value.

Table 23. Best Value Achieved by Traditional GA at Different Problem Sizes

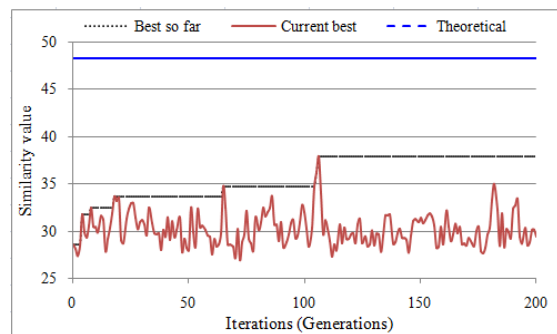
		Problem Size					
		10	20	30	40	50	100
Best value achieved by the traditional GA.		8.96	16.02	23.23	31.04	38.65	73.85
Time in milli-second	By traditional GA	5572	8706	14835	18261	10008	16678
	By GGAM	10	1	1	2	4	20

Table 24. Time Taken by Traditional GA and GGAM over 20000 Iterations at Different Problem Sizes

		Problem Size					
		10	20	30	40	50	100
Time in seconds	By traditional GA	13	16.2	16.8	18.7	18.8	27.0
	By GGAM	20	27.0	46.1	76.7	118	477.5



(a) Convergence of GGAM in the first 200 iterations.



(b) Convergence of traditional GA in the first 200 iterations.

Figure 28. The Convergence of Hybridized GA versus Traditional GA in the First 200 Iterations, n=50

Figure 30 shows the convergence behavior of the three algorithms over the first 200 iterations. It also confirms the results obtained in Figure 28, where GGAM converges to the optimal solution after around 70 iterations, while the two traditional GA algorithms are still far behind.



Figure 29. GA versus GGAM Algorithm, Matching Accuracy, Precision and Recall, Case Study 1

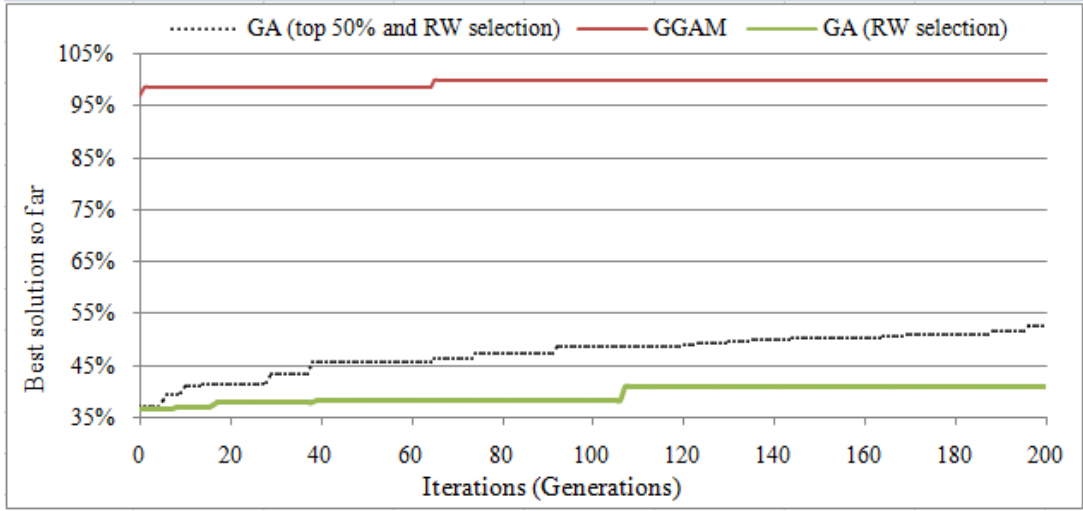


Figure 30. Convergence Behavior of Traditional GA versus GGAM over the First 200 Iterations

8.7 Empirical Validation of the Matching

In this section we investigate the performance of three different algorithms, GGRM, GGAM, and GSAM, respectively presented in Sections 6.2.1, 6.2.2, and 6.2.3, for matching UML class diagrams based on their lexical, internal, neighborhood similarity, and a combination of them. The performance of the metrics has been investigated and compared over 7 class level similarity metrics (5.3 and 5.4) and under both equal and calibrated weight settings for the compound metrics.

Experiment 4: Evaluating the matching algorithms against the similarity metrics.

Experimental objectives: Our validation of the matching algorithms has multifold objectives. First, it validates the findings of the experiments conducted in the comparison phase regarding weight calibration of the compound metrics. Second, it compares the performance of the different matching algorithms, across the different metrics, under equal and calibrated weight settings, and using within and across domains experimental

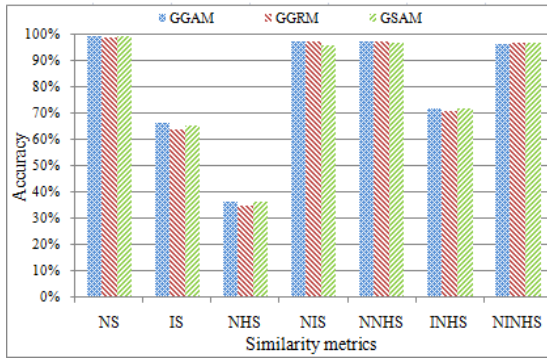
objects. Third, it provides an insights for the further activities in our consolidation framework.

Methodology: For each pair of models in each case study, and for each similarity metric, the three matching algorithms were run under equal (even) and calibrated weight settings. The matching threshold were set into 0.80. For the calibrated weight settings, the weights for the compound metrics were set as shown in Table 11, where the calibrated weights are set to the weights that give the best accuracy as suggested by the weight calibration experiments. The matching accuracy and time are reported and compared for the three algorithms.

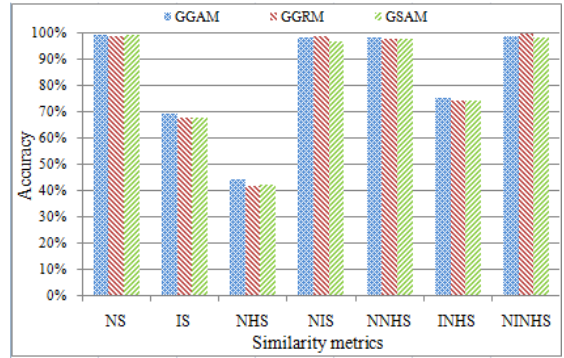
Experimental objects: Case study 1, 2.

Results and analysis:

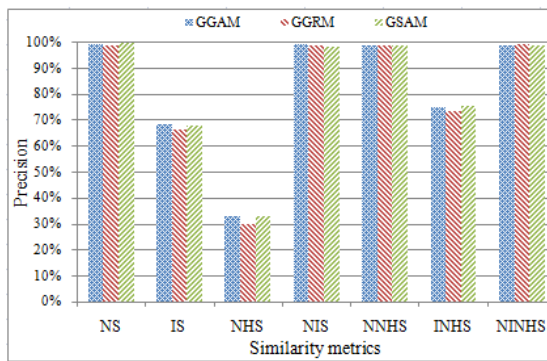
Figure 31 and Figure 32 show the matching accuracy as measured by the thee different accuracy measures, accuracy (at the top of the figures), precision (the second row in the figures), and recall (at the bottom of the figures). The figures also show the matching accuracy under both Equal weight assignment (left side of the figures) versus Calibrated weight assignment (right side of the figures) of the compound metrics. From these figures we can notice the following.



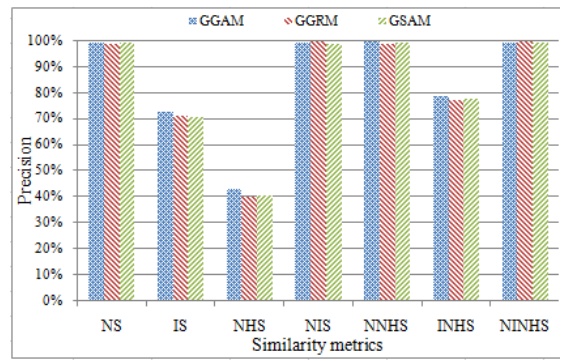
(a) Equal weight



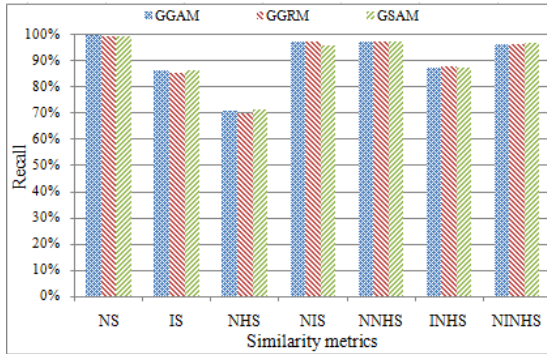
(b) Calibrated weights



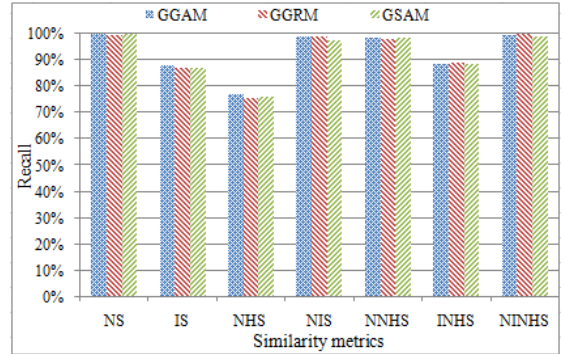
(c) Equal weight



(d) Calibrated weights



(e) Equal weight



(f) Calibrated weights

Figure 31. Matching Accuracy, Precision and Recall of GGRM, GGAM, and GSAM, Case Study 1

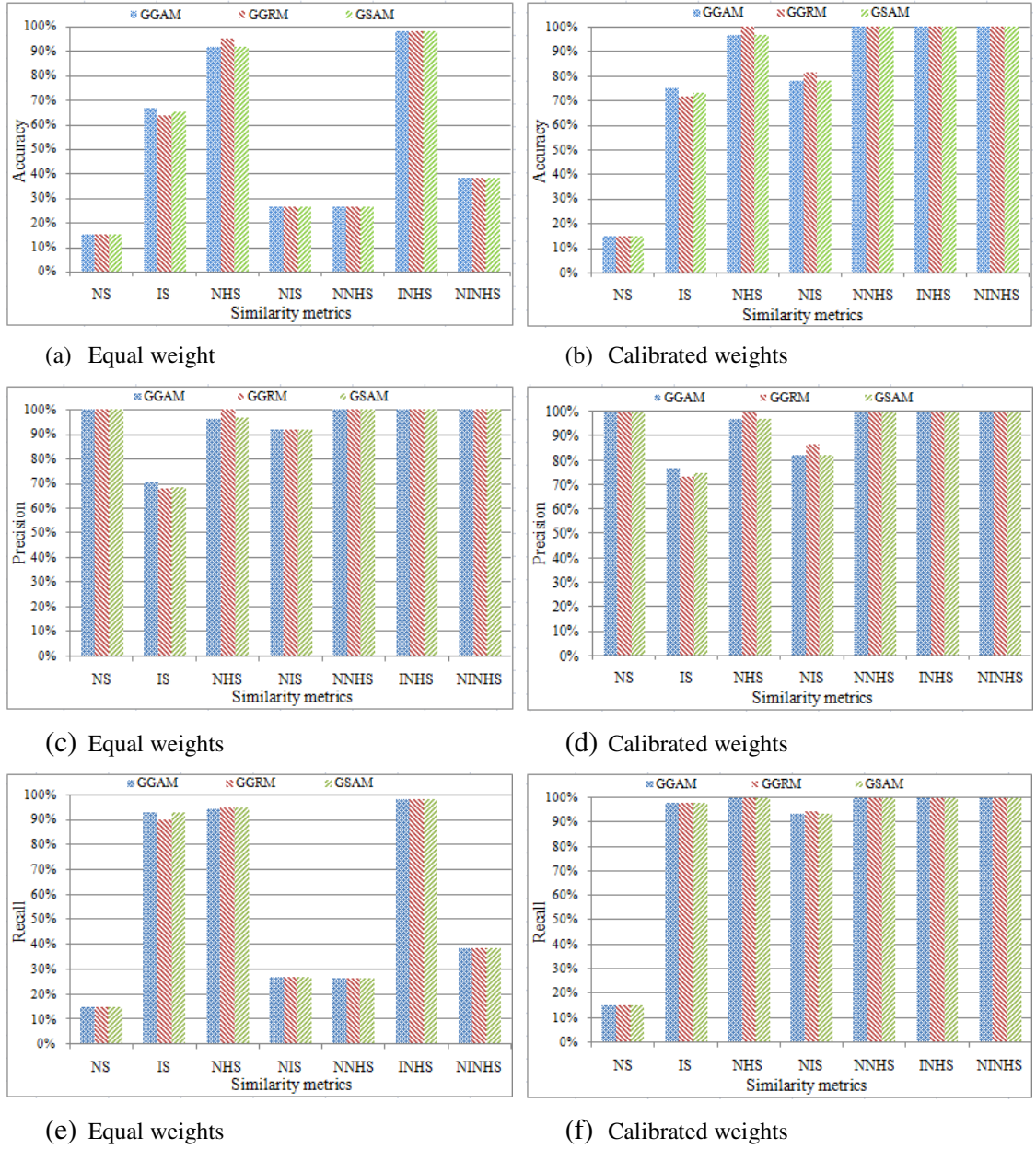


Figure 32. Matching Accuracy, Precision and Recall of GGRM, GGAM, and GSAM, Case Study 2

1. *Metrics performance (lexical versus structural)*: as it is clear from Figure 31, in Case Study 1, and under both even and calibrated weight assignments, the high precision, recall, and accuracy are achieved when the *NS* metric is present, either as a single metric or as part of a combination with other metrics. This is not surprising, for the matched models are within the same domain where the high

lexical naming similarity is expected. The worst accuracy is obtained with the structural based metric, i.e. *NHS* metric. This poor performance of *NHS* can be explained as follows. Looking at the corresponding precision results, Figure 31-(c), we can see that *NHS* shows relatively low precision, which indicates a high false positive rate (see Equation (17)), which, in turn, indicates that some dissimilar classes may have similar neighborhoods. This similarity in the neighborhood may result in identical similarity values for dissimilar classes, which represents a confusion for the matching algorithm, which ultimately results in a poor accuracy.

The situation is different with Case Study 2 (Figure 32), where the structural-based metric, *NHS*, is the superior. This is due to the fact that models across different domains (which is the case in Case Study 2) have different ontologies, and thus relying on the lexical based metric (i.e. *NS* or *IS*) only may not capture their real similarity, even if they are structurally similar.

Also, under the even weight assignment, it is clear from the two figures that while the *NS* metric is dominating the compound metrics (*NIS*, *NNHS*, and *NINHS*) in Case study 1 toward increasing the true positives (hence increasing the values of the three accuracy measures), its domination in Case study 2 is toward increasing the false negatives (hence decreasing the recall and accuracy). This is why the superiority of *NHS* in Case Study 2 is not reflected that much in the compound metrics *NNHS* and *NINHS*, under the even weight assignment, but it is clearly reflected under the calibrated weight assignment for the four metrics where the

NHS metric is present either as a single metric or as part of a combination with other metrics.

2. *Metrics performance (single versus combined information)*: the inconsistent performance of the two metrics, *NS* and *NHS*, across the two case studies, showed the limitations and the short insight of those metrics. The metrics are based on limited source of information, as the former metric is only based on the lexical naming information while the later is based on the neighborhood information alone. The *IS* metric, which is based on the internal information of the class, shows almost consistent performance across the two case studies and under both even and calibrated weight assignment. However its performance is limited in terms of the accuracy.

As per the four metrics (*NIS*, *NNHS*, *INHS*, *NINHS*), which are based on more than one type of similarity information, the results show that across the two case studies, the two compound metrics *NNHS* and *NINHS* reported high and consistent accuracy under the calibrated weight assignment. However, the performance of the two metrics *NIS* and *INHS* is not consistent across the two case studies. In Case Study 1 the *INHS* metric showed an accuracy of around 75%, under the calibrated weight assignment for its constituents (see Figure 31-(b)). This is relatively low accuracy as compared to its reported accuracy in Case Study 2 (100%), see Figure 32-(b). The reason for the low accuracy reported in Case Study 1 can be attributed to the fact that the confounding effect of the generic methods or attributes, or of the empty methods' list or attributes' list, if comes together with the similarity of the neighborhood for some classes, can lead

to increasing the false positives (hence decreasing the precision and accuracy). This reason can be witnessed by the corresponding precision result which indicates relatively high false positives.

As per *NIS*, the metric is based on two lexical metrics, *NS* and *IS*. The lexical similarity across domains (which is the situation in Case Study 2) is expected to be low, leading to a decrease in the accuracy. However, under the calibrated weight assignment, both metrics, *NIS* and *INHS*, are performing better than their constituents, across the two case studies. This shows the importance of considering different aspects of similarity information.

3. *Weight Calibration of the compound metrics*: When comparing the matching accuracy under equal versus calibrated weight assignments of the constituents of the compound metrics, the results do report an improvement in the matching accuracy. However, as shown in the two figures, this improvement may vary from a metric to another, and from a case study to another. It is clear from Figure 32 that, under the equal weight assignment, the low accuracy was obtained with the compound metrics *NIS*, *NNHS*, and *NINHS*. The *NS* metric is one of the constituents in each of these metrics. Thus, under the equal weight assignment the *NS* metric dominates the three compound metrics toward increasing the false negative. This last claim can be observed if we look at the result of the accuracy measure (Figure 32-(a)) in the light of the both the precision (Figure 32-(c)) and recall (Figure 32-(e)). Since the recall for these three metrics is low, under the even weight assignment, it means that the false negatives reported by these metrics is high, see Equation (18). The high false negative rate, companied with

low false positives, is an indication that high rate of the matched classes could not pass the matching threshold, which can happen due to the low similarity scores. Under the calibrated weight assignments of the constituents of the compound metrics, the undesirable domination of the *NS* is controlled and each constituent is assigned a weight that makes its contribution best towards decreasing the false negatives and false positives, and hence increasing precision, recall, and accuracy. This is clearly depicted in the matching accuracy result reported under the calibrated weight assignment, the right hand side of Figure 32, where the matching precision and accuracy show high improvement over the results obtained under the equal weight assignment for the three compound metrics. This improvement resulted in an accuracy of 100% for the 3 compound metrics (*NNHS*, *INHS*, and *NINHS*). However, the max accuracy we obtain for the *NIS* metric was around 81%. This emphasizes the importance of structural information for similarity assessment across domains, as we obtained only limited accuracy even under the calibrated weights.

This is similar to what happens with *INHS* metric in Case Study 1, as the best accuracy obtained when not including the class name similarity is around 75%, which emphasizes the importance of the class name information for the similarity assessment within the domain

In Case Study 1, the improvement in the matching accuracy achieved under the calibrated weights of the constituents of the compound metrics was not that much over the accuracy reported under the equal weight assignment. Under the calibrated weight, the compound metric *NHS* reported an accuracy of 10% higher

than its reported accuracy under the equal weight assignment for its constituents. However, even with this improvement, its performance is still limited within the domain, as explained earlier. For the metrics *NIS*, *NNHS*, and *NINHS*, only little improvement is achieved and this can be explained as follows. Referring to Figure 20-(c), we can see that when the w_n (the weight coefficient of *NS* metric) is assigned any non-zero weight, we usually get high accuracy with slight difference from a non-zero assignment to another. This is due to the domination of the *NS* metric under any non-zero weight for its weight coefficient, as explained earlier. Additionally, the performance of *IS* and *INHS* shows improvement under the calibrated weight as compared to their performance under the equal weights for their constituents. However, these two metrics are missing the lexical naming similarity of the class which is important source of similarity information within the domain. This is why their accuracy is still limited even under equal weight assignment.

4. *Performance of the matching algorithms*: the performance of the three algorithms (GGRM, GGAM, and GSAM) was evaluated over the different metrics and across the two case studies in terms of both the matching accuracy and time. As shown in Figure 31 and Figure 32, the three algorithms reported competitive performance against each other in terms of precision, recall and accuracy. We cannot claim absolute winner, but if we count the number of times where each algorithm is performing better than the others, we can say that in Case Study 1 GGAM is performing slightly better than both GGRM and GSAM. However, the difference in the accuracy between the different algorithms is within 1% to 3%.

It is worth recalling here that the matching problem has two facets, the accuracy of the similarity assessment and the complexity the matching algorithm. For example, the *NS* metric in Figure 32-(a) reported a low accuracy of around 15%. In the light of the recall (Figure 32-(e)) and the principles of the GGRM algorithm^{*}, it is clear that the problem of this low accuracy is coming from the metric facet of the problem, which reported low similarity values (not able to pass the threshold). On the other hand, the problem with *IS* metric is different. It is coming from both metric facet and algorithm facet. The metric problem can be inferred from the precision results which indicates relatively low precision, which in turn indicates relatively high false positives. The high false positives means that high similarity values were assigned for dissimilar classes making the wrong match able to pass the threshold. This misleading similarity values can be due to confounding effect of generic attributes and methods, where the similarity assessment of two different, but internally identical, classes in a model against other classes in the other model can lead to identical similarity values for the *IS* metric. If these identical values are encountered by the matching algorithms as the highest values, the algorithm will do matching with the first value it encounters, and the first to be encounter may differ from an algorithm to other algorithm. This may result in a performance difference among the different algorithms. This is again emphasizing the importance of an accurate similarity assessment.

Time wise, Table 25 shows the average run time, with the standard deviation, of the three algorithms against different size of the *ES* matrix. It is clear from the

^{*} Greedy approach usually struggles for high values

table that the algorithms do report great differences in their running time. The GGRM algorithm is deterministic, simple, straightforward, and the final solution is produced in a single cycle. Solutions of GSAM and GGAM go over different optimization cycles (generations), making the running time proportional to the number of generations. Additionally, GGAM is a population based algorithm, which means that in each generation it works with many solutions at the same time. This is expected to increase the iteration run time in proportion with the population size. This is why the highest run time was reported by the GGAM and the lowest run time was reported by the GGRM.

Figure 33 also compares the matching accuracy of the three different algorithms, across the different similarity metrics, using the artificial (generated) data, with perturbation settings as indicated in Table 22. The results confirms to the above discussion.

Table 25. Matching Time Taken by GGRM, GGAM, and GSAM Algorithms for Each Pair of Models

	M_0M_1	M_0M_2	M_0M_3	M_0M_4	M_1M_2	M_1M_3	M_1M_4	M_2M_3	M_2M_4	M_3M_4
Time, in Seconds, Avg.	0.001	0.002	0.002	0.001	0.003	0.002	0.002	0.003	0.003	0.003
taken by GGRM Std.	0.001	0.002	0.002	0.002	0.006	0.002	0.002	0.003	0.003	0.001
Time, in Seconds, Avg.	0.949	0.972	0.977	1.133	1.239	1.176	1.187	2.314	2.193	2.229
taken by GGAM Std.	1.061	1.117	1.112	1.271	1.352	1.347	1.298	2.558	2.446	2.581
Time, in Seconds, Avg.	34.521	34.310	34.303	38.346	42.565	41.031	40.871	75.765	76.354	76.935
taken by GSAM Std.	36.754	38.653	38.574	43.487	46.606	44.478	44.668	83.105	84.994	85.62
Size of ES matrix	49 x 53	49 x 71	49 x 71	49 x 67	53 x 71	53 x 71	53 x 67	71 x 71	71 x 67	71 x 67

To sum up, the evidences reported from our different experiments for element to element matching suggest the following findings:

- (a) Relying on a single metric may not usually lead to an accurate match between the elements of two models.

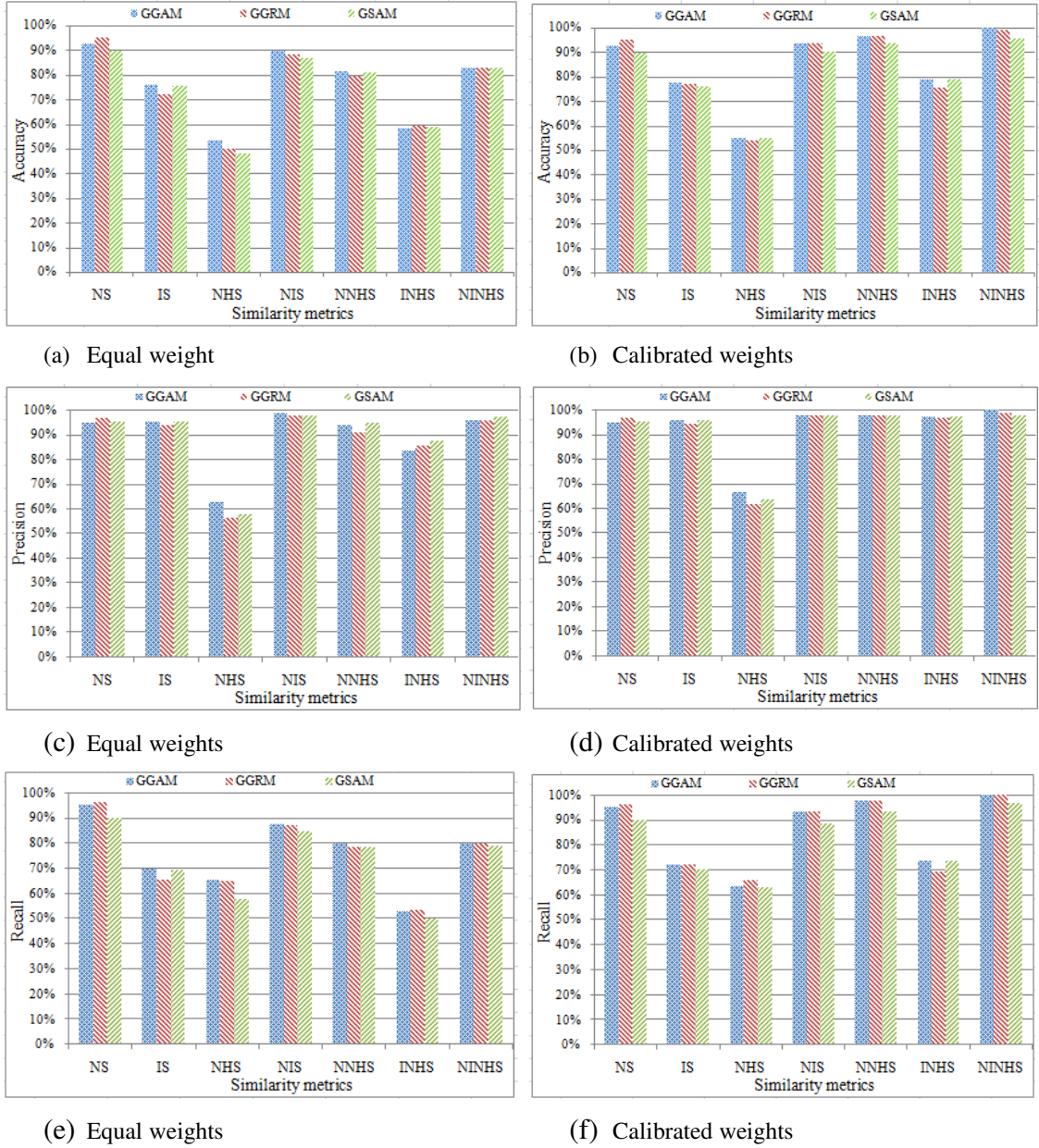


Figure 33. Matching Accuracy of GGRM, GGAM, and GSAM, Case Study 3

(b) The weights assigned to the individual metrics, constituting a compound metric, is crucial in calibrating the actual contribution of each constituent.

(c) Metrics based on multiple source of information showed better overall accuracy than do those with single source of information, under the

appropriate weight assignment which makes the contribution of each source of information more convenient based on the context of measurement.

- (d) The competitive performance of the three matching algorithms over the different metrics, in terms of the accuracy, can be considered as group voting about the soundness or the limitation of the metrics' performance; and it can also be considered as validation mechanism for the matching performance of the algorithms themselves; additionally, it makes the selection of one over the other as context based choice.

8.8 Empirical Investigation of the Consolidation and the Ruse of the Reference Model

This investigation has a twofold objective. First, it provides a proof of concept for the proposed staged merging algorithms. Second, we investigate the overhead and the reuse potential provided by the reference model over different points of time. The investigation was conducted as follows.

Experiment 5: Building the reference model

Experimental objectives: This experiment has twofold objective: 1) to validate the merging algorithms for building the reference model given the pair-wise matching similarity matrices *MSMs*, 2) to show that the unrelated models will be filtered out and the reference will be built based on the majority of the instances.

Experimental objects: Case Study 0, and Case Study 2.

Methodology: Six input instances are given as input models. Four of these instances are coming from Case Study 0, and two instances are coming from case study 2. The pair-

wise similarity between the elements of each pair of models was evaluated. Weights are set guided by the findings of Experiments 2 and 3 (Section 8.5). The matching was performed to produce the *MSM* matrix for each pair of input models. Using the similarity information in the *MSM* matrices, unrelated models are filtered first as explained in Section 4.4.5. Then the remaining models are generalized, as described in Sections 7.4 & 7.5, to build the reference model.

Table 26, shows the pair-wise models' similarity. As shown in the table, the low similarity values in the last two columns indicate the dissimilarity between the two models M_4 and M_5 with each of the other models, M_0 through M_3 . To filter out the unrelated models, the tool computes the average similarity of each model to the others (see Table 27), and the model with the lowest average similarity under a threshold of 70% will be filtered out. Despite the high similarity between M_4 and M_5 , the average similarity of each one of them with the other models is lower than the average similarity of any one of the other models. Since M_4 has the lowest average similarity with the other models, it is filtered out, and the average similarity of each model with the others is recomputed for the remaining models, without M_4 . Table 28, shows the average similarity of each model with the other models, after filtering out M_4 . As we can see, in this table, the average similarity of each of the models M_0 through M_3 to the others increased, signifying that the models become more cohesive after removing M_4 . On the other hand, the average similarity of M_5 (the most similar one to M_4) with the other models decreased after filtering out M_4 , signifying its heterogeneity to the other models in the set, and its homogeneity with the already filtered model (i.e. M_4).

Table 26. Pair-wise Models' Similarity between 6 Input Models

	<u>M₀</u>	<u>M₁</u>	<u>M₂</u>	<u>M₃</u>	<u>M₄</u>	<u>M₅</u>
M₀	1	0.85	0.76	0.9	0.22	0.28
M₁		1	0.84	0.83	0.15	0.14
M₂			1	0.77	0.13	0.13
M₃				1	0.15	0.14
M₄					1	0.87
M₅						1

Table 27. The Average Similarity of Each Model to the Other Models

<u>Model</u>	<u>M₀</u>	<u>M₁</u>	<u>M₂</u>	<u>M₃</u>	<u>M₄</u>	<u>M₅</u>
Avg. Similarity to other models	0.60	0.56	0.53	0.56	0.30	0.31

Table 28. Pair-wise Models' Similarity After Removing M₄

<u>Model</u>	<u>M₀</u>	<u>M₁</u>	<u>M₂</u>	<u>M₃</u>	<u>M₅</u>
Avg. Similarity to other models	0.70	0.67	0.63	0.66	0.17

Table 29. Pair-wise Models' Similarity After Removing M₄ and M₅

<u>Model</u>	<u>M₀</u>	<u>M₁</u>	<u>M₂</u>	<u>M₃</u>
Avg. Similarity to other models	0.84	0.84	0.79	0.83

Table 29 shows the average similarity of each of the models M_0 through M_3 to the others after filtering out M_4 and M_5 . Again, as shown in the table, removing model M_5 from the set makes the remaining models more cohesive and the average similarity of each one of them to the others increased. Also, the average similarity of each model to the others becomes more than the filtering threshold, which means no more filtering, and the algorithm will go ahead to generalize all the four remaining models to build the reference.

Snapshots of the reference model catalog for models of Case Study 0 is shown in Figure 34. As demonstrated in the catalog, class names are kept as aliases, for the sake of instantiation, and the name for the reference class can be chosen in different ways. It can be the most frequent name among instance classes, the least common concepts, or simply any of the names appearing in one of the generalized instances. We opted to go with the last option. The figure also shows that, attributes (also methods) that appears in some instances but not in the others are tagged with a vector indicating in which instance(s) this attribute shows up (marked with 1) and in which it does not (marked with 0). We call this vector an instance tag. The length of the instance tag depends on the number of generalized instances, which may make it too long if the number of instances is large in the reference. However, the algorithm can be configured in such a way that if the number of instances reach a certain number, a percentage of the attribute frequency over the different instances will be shown instead. Attributes (also methods) that are not tagged with the instance tag means that they are common among all the instances represented by that class. Relationships are also tagged by an instance tag indicating the occurrence of the relationship at the different instances generalized by the reference model. Relationships with variation points are indicated in the relation instance tag by the letter “v” while the letter “c” indicates that the relationship is between two common classes. The letter “o” in the instance tag indicates that the relationship is with an optional class. Snapshots of the reference model catalog for models of Case Study 2 is shown in Figure 35.

```

=====Reference Model Catalog=====

=== Reference Common Classes===

Class : Airplane <Alias:Airplane ; Airplane ; Plane ; Plane ; >
Attributes:
  code : String
  model : String
  status : String
Methods:

Class : Airway <Alias:Airway ; Airline ; Airline ; Airway ; >
Attributes:
  address : String
  name : String
  <opt><1:1:0:1>route : String
  <opt><1:0:1:1>type : String
Methods:
  addroute(String:routeDescription)
  <opt><1:1:0:0>cancelRoute(Integer:routeCode)
  <opt><0:0:0:1>removeRoute(Integer:routeNumber)

Class : Reservation <Alias:Reservation ; Booking ; Reservation ; Booking ; >
Attributes:
  date : String
  number : Integer
  <opt><0:1:0:0>validity : Integer
Methods:
  confirm(Integer:number) : Boolean
  cancel(Integer:number) : Boolean

Class : Airdrome <Alias:Airdrome ; Airport ; Airport ; Airdrome ; >
Attributes:
  name : String
  city : String
  <opt><1:0:0:0>code : String
  <opt><1:1:0:0>country : String
  <opt><1:0:0:0>description : String
  <opt><0:1:1:1>type : String
Methods:

Class : Traveler <Alias:Traveler ; Passenger ; Traveler ; Traveler ; >
Attributes:
  <opt><1:1:0:0>forename : String
  <opt><1:1:0:0>surname : String
  mobile : String
  <opt><1:1:0:0>fax : String
  <opt><1:1:1:0>passportNumber : Integer
  age : Integer
  <opt><0:0:1:1>name : String
  <opt><0:0:1:0>phoneNumber : Integer

=====Inter-Common relations...=====
<c1:1:1:1> Between Class Reservation(ownedAssociation) and Traveler(ownedAssociation) Relation Name:makes
<c1:1:1:1> Between Class Airplane(ownedAssociation) and Airway(memberAssociation) Relation Name:own
=====Common-Variant relations...=====
<v0:0:0:1> Between Class Airway(ownedAssociation) and Flight(ownedAssociation) through VP0 Relation Name: define
<v0:0:1:1> Between Class Airway(ownedAssociation) and Flight(ownedAssociation) through VP0 Relation Name: offer
<v0:0:1:0> Between Class Airway(ownedAssociation) and Flight(ownedAssociation) through VP0 Relation Name: schedule
<v1:1:0:0> Between Class Airway(memberAssociation) and Offered Flight(ownedAssociation) through VP0 Relation Name: offer
<v1:1:0:0> Between Class Airway(ownedAssociation) and Scheduled Flight(ownedAssociation) through VP0 Relation Name: define
<v0:0:1:0> Between Class Airdrome(ownedAssociation) and Flight(ownedAssociation) through VP0 Relation Name: stopover
<v0:0:1:1> Between Class Airdrome(ownedAssociation) and Flight(ownedAssociation) through VP0 Relation Name: arrival
<v0:0:1:1> Between Class Airdrome(ownedAssociation) and Flight(ownedAssociation) through VP0 Relation Name: depature
<v1:1:0:0> Between Class Airdrome(ownedAssociation) and Scheduled Flight(ownedAssociation) through VP0 Relation Name: stopover
<v1:1:0:0> Between Class Airdrome(ownedAssociation) and Scheduled Flight(ownedAssociation) through VP0 Relation Name: arrival
<v1:1:0:0> Between Class Airdrome(ownedAssociation) and Scheduled Flight(ownedAssociation) through VP0 Relation Name: depature
<v0:0:1:1> Between Class Airplane(memberAssociation) and Flight(ownedAssociation) through VP0 Relation Name: assignedto
<v1:1:0:0> Between Class Airplane(memberAssociation) and Offered Flight(ownedAssociation) through VP0 Relation Name: assignedto
<v0:0:1:1> Between Class Reservation(ownedAssociation) and Flight(ownedAssociation) through VP0 Relation Name: concern
<v1:1:0:0> Between Class Reservation(ownedAssociation) and Offered Flight(ownedAssociation) through VP0 Relation Name: concern

=====Common-Optional relations...=====
<o0:1:1:0> Between Class Airdrome(memberAggregation) and Terminal(ownedAssociation) through OP0 Relation Name: has

=====Intra-Variant relations...=====
=====Inter-Opt relations...=====

```

Figure 34. Snapshots from the Reference Model Catalog, Case Study 0.


```

*****Reference Model Catalog*****

== Reference Common Classes==

Class : Employee <Alias:Employee ; Employee ; Employee ; Employee ; >
Attributes:
  name : String
  employeeNumber : String
Methods:

Class : RepairTechnician <Alias:RepairTechnician ; Spec Employee ; DiagnosisDoctor ; AdmissionsOfficer ; >
Attributes:
Methods:

Class : ReceptionTechnician <Alias:ReceptionTechnician ; worker ; PrimaryDoctor ; Faculty ; >
Attributes:
Methods:

Class : Estimate <Alias:Estimate ; Evaluation ; Diagnosis ; Admission ; >
Attributes:
  number : Integer
  date : String
Methods:

Class : Customer <Alias:Customer ; Guardian ; Guardian ; Guardian ; >
Attributes:
  name : String
  address : String
Methods:

Class : ComputerRepairShop <Alias:ComputerRepairShop ; Institution ; Hospital ; University ; >
Attributes:
  name : String
  location : String
Methods:

Class : RepairLog <Alias:RepairLog ; StayHistory ; MedicalHistory ; Transcript ; >
Attributes:
  startDate : String
  endDate : String
Methods:

Class : RepairEvent <Alias:RepairEvent ; StayRecord ; HospitalStay ; SemesterTranscript ; >
Attributes:
  number : Integer
  date : String
Methods:

Class : ComputerShopChain <Alias:ComputerShopChain ; Institution Group ; HospitalConsortium ; UniversitySystem ; >
Attributes:
  name : String

Class : Computer <Alias:Computer ; Applicant ; Patient ; Student ; >
Attributes:
  <opt><1:0:0>serial : Integer
  <opt><1:0:0>manufacturer : String
  <opt><0:1:1:1>name : String
  <opt><0:1:0:1>number : String
  <opt><0:0:1:0>SSN : String
Methods:

=====Variants=====

=====Optionals=====

=====Inter-Common relations.....=====

<c1:1:1:1> Between Class ComputerRepairShop(ownedAssociation) and Computer(ownedAssociation) Relation Name:
<c0:1:0:1> Between Class ReceptionTechnician(ownedAssociation) and RepairEvent(ownedAssociation) Relation Name:Advises
<c1:0:1:0> Between Class ReceptionTechnician(ownedAssociation) and RepairEvent(ownedAssociation) Relation Name:InChargeof
<c1:1:1:1> Between Class Employee(generalization) and ReceptionTechnician(specialization) Relation Name:
<c1:1:1:1> Between Class RepairTechnician(ownedAssociation) and Estimate(ownedAssociation) Relation Name:Responsiblefor
<c1:1:1:1> Between Class Customer(ownedAssociation) and Computer(ownedAssociation) Relation Name:Responsiblefor
<c1:1:1:1> Between Class ComputerRepairShop(ownedAssociation) and ComputerShopChain(memberAggregation) Relation Name:
<c1:1:1:1> Between Class Employee(generalization) and RepairTechnician(specialization) Relation Name:
<c1:1:1:1> Between Class RepairLog(memberAggregation) and RepairEvent(ownedAssociation) Relation Name:
<c1:1:1:1> Between Class RepairLog(ownedAssociation) and Computer(ownedAssociation) Relation Name:
<c1:1:1:1> Between Class Employee(ownedAssociation) and ComputerRepairShop(ownedAssociation) Relation Name:workat

=====Common-Variant relations.....=====

=====Common-Optional relations.....=====

=====Intra-Variant relations.....=====

=====Inter-Opt relations.....=====

```

Figure 35. Snapshots from the Reference Model Catalog, Case Study 2

Experiment 6: Reference reuse.

Experimental objective: This experiment has twofold objective. First, it provides more validation for the merging algorithms for building the reference model, given the pair-wise matching similarity matrices *MSMs*. Second, it shows the evolution of the reference model along with its reuse potential.

Experimental objects: Case Study 1, which consists of 5 models, and Case Study 3, which consists of 10 input models generated by the instance generator as described in 8.2, with perturbation parameters as depicted in Table 22.

Methodology: The generalization is performed as described in Experiment 5. Because of the randomness involved in both the instances generation and the selection process when generalizing the models, the experiment results are repeated over five runs.

Results and analysis:

Results show that Case Study 1 and Case Study 3 share similar patterns. To make the analysis smooth and concise, our discussion will be mainly focusing on the results reported based on Case Study 3. Should there be something special about Case Study 1, we will mention it explicitly. Otherwise, the corresponding figures and tables of the results reported based on Case Study 1 should be sufficient to show the trends with regard to Case Study 1 in the light of the discussion about Case study 3.

Figure 17 shows the trace matrix of class distribution over the different 10 generated instances, numbered from 0 to 9, for one run, out of five runs, representing Case Study 3. As we can see in this figure, some classes are present in all the instances, some exist in some instances but not in the others, while other classes show up only in one instance.

Figure 36 shows the number of reference common classes in the reference model as new instances are added to the reference. As we can see in this figure, as new instances are added to the reference, the number of common classes decreases. This is expected, because the common class (according to Definition 7.6) must represent all the instances consolidated in the reference so far. If a common class in the reference model does not have a commonality with a class in the new instance the similarity level of the common class is changed by the merging algorithm from common into optional. Hence, the number of common classes in the reference model is monotonically a decreasing function of the number of instances added to the reference model. Figure 37 shows the same trend with regard to Case Study 1.

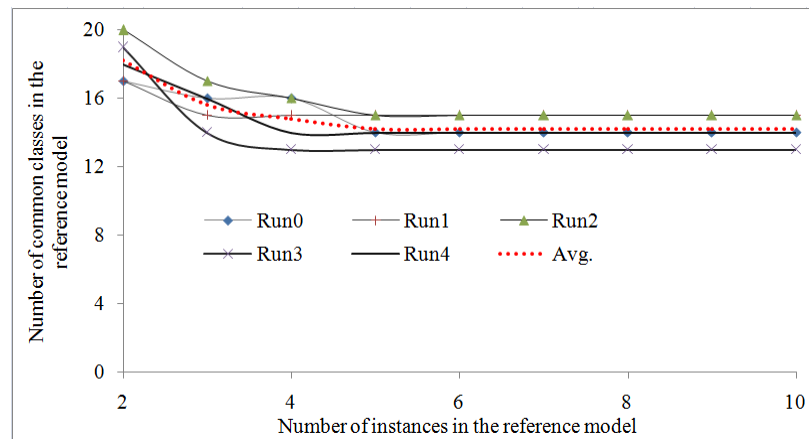


Figure 36. Reference Common Classes Evolution as More Instances Are Added to the Reference, Case Study 3

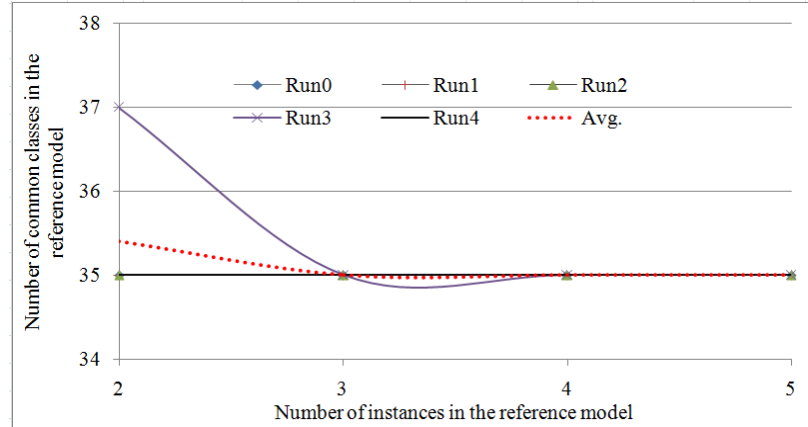


Figure 37. Reference Common Classes Evolution as More Instances Added to the Reference, Case Study 1

Figure 38 shows optional points creation, evolution, and reuse as input models are generalized to the reference model. The figure shows a snapshot (colored in different grayscales for demonstration purpose) of the reference model optional points after 10 instances have been generalized. A trace for this run showed that the models were generalized in the order of M_7 and M_8 first, then M_2 , M_0 , M_1 , M_6 , M_9 , M_3 , M_5 , M_4 , in order, one at a time. Optional points shaded in *dark gray* (e.g. OP0) are created due to the appearance of classes, in a new generalized instance, that have no commonality with any optional class in the reference. These classes are also shaded in the figure in dark gray, to demonstrate that the corresponding optional points are created due to these optional classes. Optional points created due to the similarity level conversion (e.g. from common to optional) are shaded in *gray*, e.g. OP14. The optional class in the new instance generalized under an existing optional point is shaded in *light gray*, e.g. optional class o1-7 in the column M_0 . Following this tracing guide, we can see that 14 optional points were created when generalizing the first pair, i.e. M_7 and M_8 . The reason is that M_7 and M_8 are the first pair, randomly picked by the algorithm, to be generalized. Any optional class in one of them means that it does not exist in the other, otherwise it would not be an

optional. Therefore, an optional point is created for each optional class. Since the algorithm found that 6 classes exist in M_7 but not in M_8 , and 8 classes exist in M_8 but not in M_7 , the algorithm created 14 optional points, one for each optional class. Figure 39 shows a prior version of the optional points in the reference model, when only two instances (M_7 and M_8) are in the reference.

When generalizing model M_2 , the algorithm found that 5 classes of model M_2 have commonality with five optional classes in the reference. Hence each class of the five classes is generalized under the corresponding optional point, shaded in the figure as light-gray. The algorithm also found that four of the common classes in the reference does not exist in M_2 , which entailed changing their similarity level into optional. This is why the algorithm created the optional points OP_{14} through OP_{17} , shaded in gray. Additionally, 4 classes of M_2 have no commonality with any class in the reference, resulting in a creation of 4 more optional points (OP_{18} through OP_{21}). Thus, the total number of optional points created due to the generalization of M_2 is 8. Hence, the Total number of optional points in the reference after generalizing M_2 becomes $14+8=22$ optional points.

	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9
OP0	o1-7	o1-7	o1-6	null	o1-4	null	null	o1-5	null	o1-3
OP1	null	o1-14	o1-14	o1-15	null	o1-12	o1-14	o1-12	null	null
OP2	null	null	null	null	null	null	null	o1-13	null	o1-10
OP3	null	null	null	o1-26	o1-21	o1-24	null	o1-24	null	o1-22
OP4	o1-25	o1-27	o1-25	o1-28	o1-22	o1-26	o1-24	o1-26	null	null
OP5	null	o1-31	null	null	o1-24	null	o1-29	o1-29	null	null
OP6	null	null	null	o1-2	o1-1	null	null	null	o1-1	null
OP7	o1-3	null	null	null	o1-2	null	null	null	o1-2	o1-1
OP8	null	null	o1-13	o1-14	null	null	o1-13	null	o1-13	null
OP9	o1-14	o1-15	null	null	null	null	null	null	o1-14	o1-11
OP10	null	o1-16	null	null	null	o1-13	null	null	o1-15	null
OP11	null	o1-18	o1-16	null	o1-12	o1-16	null	null	o1-18	o1-14
OP12	null	null	null	null	null	null	null	null	o1-29	null
OP13	null	null	null	null	null	null	null	null	o1-31	null
OP14	o1-18	null	null	o1-19	o1-14	null	null	o1-17	o1-20	null
OP15	null	null	null	null	null	o1-3	o1-3	o1-2	o1-4	null
OP16	null	null	null	null	null	o1-25	null	o1-25	o1-27	o1-24
OP17	o1-15	null	null	o1-16	null	o1-14	o1-15	o1-14	o1-16	null
OP18	o1-1	o1-1	o1-1	o1-1	null	o1-1	o1-1	null	null	null
OP19	o1-5	o1-5	o1-4	null	null	o1-5	o1-5	null	null	null
OP20	null	null	o1-24	o1-27	null	null	o1-23	null	null	null
OP21	o1-27	o1-28	o1-27	null	null	o1-27	o1-26	null	null	o1-25
OP22	null	o1-17	o1-15	o1-17	null	o1-15	null	o1-15	o1-17	o1-13
OP23	null	o1-4	o1-3	o1-5	null	null	o1-4	o1-3	o1-5	null
OP24	null	o1-25	o1-23	o1-25	o1-20	o1-23	null	o1-23	o1-26	o1-21
OP25	o1-2	null	null	null	null	null	o1-2	null	null	null
OP26	o1-16	null	null	null	o1-11	null	o1-16	null	null	o1-12
OP27	o1-24	o1-26	null	null	null	null	null	null	null	o1-23
OP28	o1-26	null	o1-26	null	null	null	o1-25	o1-27	o1-28	null
OP29	null	o1-2	null	o1-3	null	null	null	null	null	null
OP30	null	o1-29	null	o1-29	null	null	o1-27	null	null	o1-26
OP31	o1-4	o1-3	o1-2	o1-4	null	o1-2	null	o1-1	o1-3	null
OP32	null	null	null	o1-12	null	o1-10	null	null	null	null
OP33	null	null	null	o1-31	null	o1-29	null	null	null	null
Keys:										
	Reusing existing OP		New OP is created			Created OP due to converted similarity level				

Figure 38. Optional Points Creation and Reuse During Generalization, Case Study 3

Optional points creation rate

Figure 40 shows the number of optional points in the reference model against the number of instances in the reference (i.e. against the reference size). As shown in the figure, the number of optional points in the reference model is monotonically increasing function of the number of instances added to the reference model. This is due to the fact that the new instance is likely to have some classes with no similarity to any class in the reference model, especially when the reference has less number of instances. However, as it is clear from the figure that as the reference has more instances, the number of optional points in the reference become almost stable and the increase in this number, if any, is slow. The reason is that as the number of instances in the reference model becomes

larger, most (if not all) of the classes of the new generalized model will have commonality with either a common or an optional class in the reference model.

	Instances									
	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9
Optional points	OP0	null	null	null	null	null	null	o1-5	null	null
	OP1	null	null	null	null	null	null	o1-12	null	null
	OP2	null	null	null	null	null	null	o1-13	null	null
	OP3	null	null	null	null	null	null	o1-24	null	null
	OP4	null	null	null	null	null	null	o1-26	null	null
	OP5	null	null	null	null	null	null	o1-29	null	null
	OP6	null	null	null	null	null	null	null	o1-1	null
	OP7	null	null	null	null	null	null	null	o1-2	null
	OP8	null	null	null	null	null	null	null	o1-13	null
	OP9	null	null	null	null	null	null	null	o1-14	null
	OP10	null	null	null	null	null	null	null	o1-15	null
	OP11	null	null	null	null	null	null	null	o1-18	null
	OP12	null	null	null	null	null	null	null	o1-29	null
	OP13	null	null	null	null	null	null	null	o1-31	null

Figure 39. Optional Points Creation During Generalization, First Pair, Case Study 3

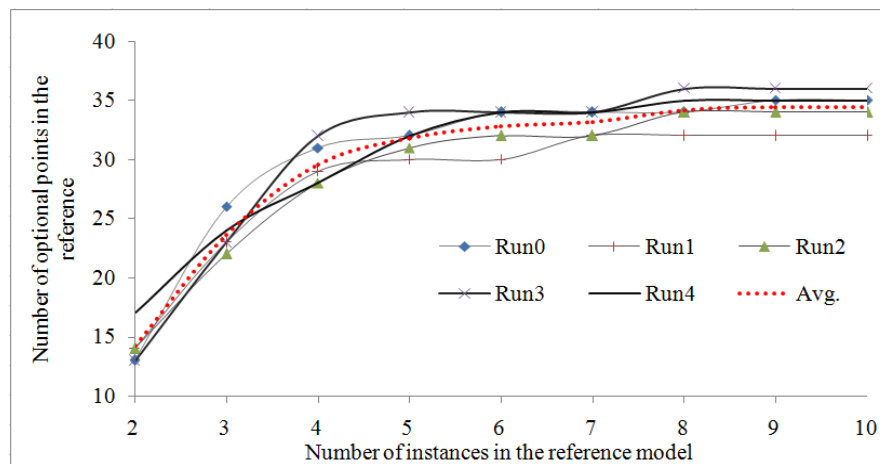


Figure 40. Optional Points versus Number of Instances in the Reference Model, Case Study 3

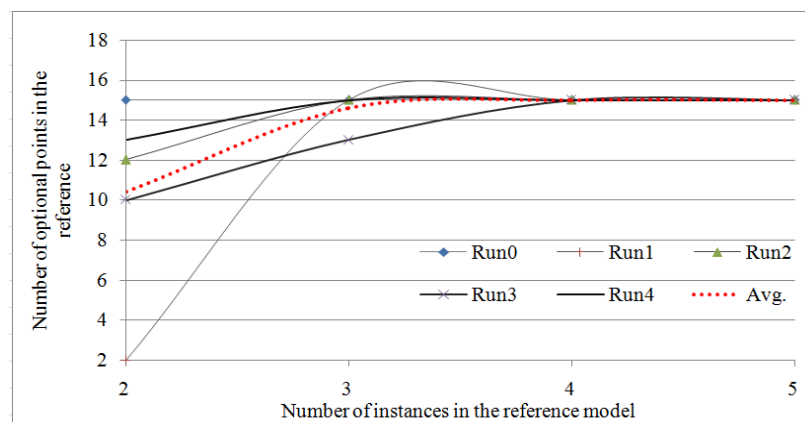


Figure 41. Optional Points versus Number of Instances in the Reference Model, Case Study 1

The results in Figure 40 is confirmed by the results in Figure 42, which show the number of optional points added due to the generalization of a new instances versus the size of the reference model. As it is clear in this figure, when there is no or few instances in the reference model the creation rate of the optional points is high.

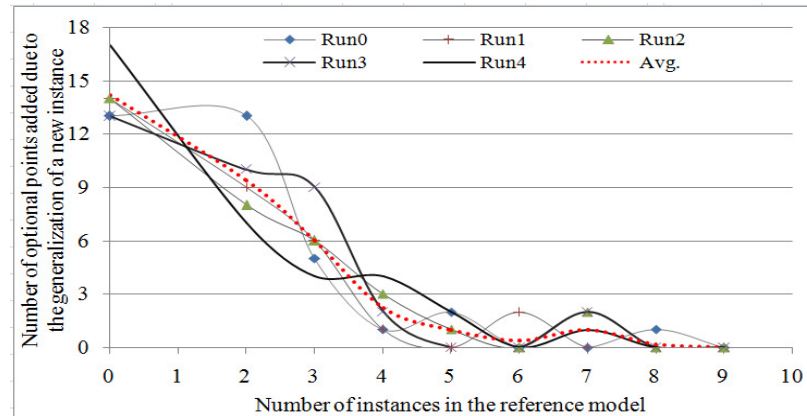


Figure 42. Number of Optional Points Added Due to the Generalization of a New Instance versus the Size of the Reference Model, Case Study 3

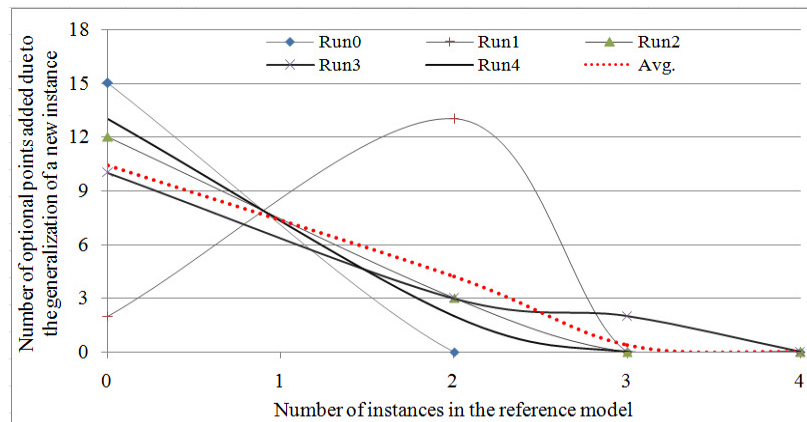


Figure 43. Number of Optional Points added Due to the Generalization of a New Instance versus the Size of the Reference Model, Case Study 1

Figure 44 also shows that when the reference model has less number of instances, high percentage of the optional points represents only a single instance. However, as the reference gets more instances, this percentage decreases into a low value. Simultaneously, the percentage of optional points generalizing more than one instance goes in the other direction of the scale, see Figure 46. This is again due to the fact that

when the reference model has less number of instances the new instance is likely to have some classes with no similarity to any class in the reference model, which results in creating new optional points, which in turn results in increasing the percentage of the single instance optional points. However, when the number of instances in the reference model becomes larger, most of the classes of the new generalized model will have commonality with either a common or an optional class in the reference model. The latter case results in increasing the percentage of optional points generalizing more than one instance.

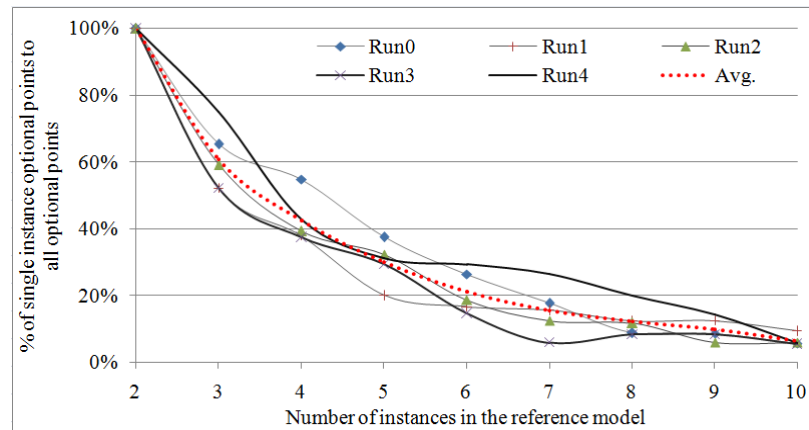


Figure 44. Percentage of Single Instance Optional Points against the Number of Instances in the Reference Model, Case Study 3

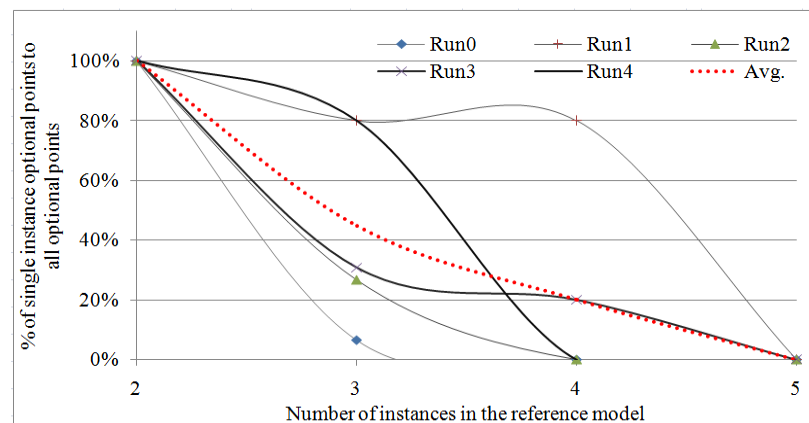


Figure 45. Percentage of Single Instance Optional Points against the Number of Instances in the Reference Model, Case Study 1

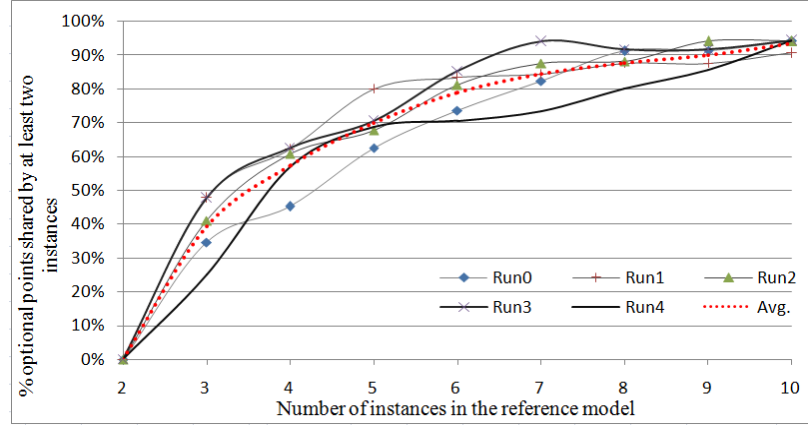


Figure 46. Percentage of Multiple Instances Optional Points against the Number of Instances in the Reference Model, Case Study 3

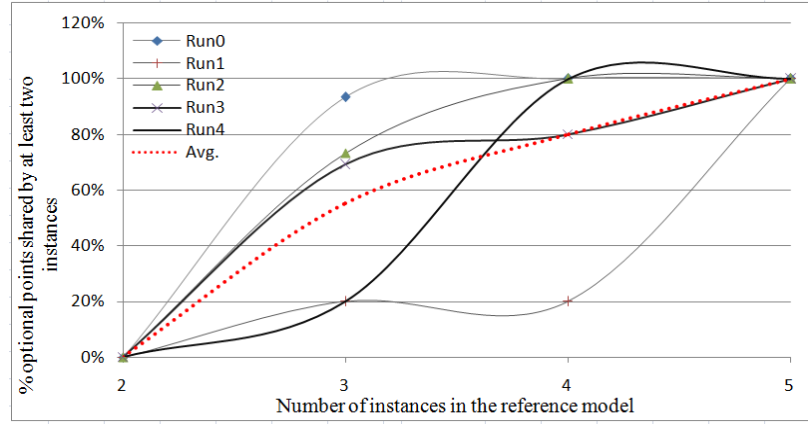


Figure 47. Percentage of Multiple Instances Optional Points against the Number of Instances in the Reference Model, Case Study 1

Reference model commonality and reuse

In the context of the software product line, commonality is a key metric that indicates reuse payoff of a feature across the SPL [150]. According to the Software Engineering Institute [151], the commonality C_F of a feature F is computed as follows.

$$C_F = \frac{\|P_F\|}{n},$$

where $\|P_F\|$ is the number of products within the SPL that use the feature, and n is the total number of products in the SPL. The metric values are between 0 and 1.

We adopted and redefined this metric to measure the reuse potential of the classes within the reference model as follows.

The commonality C_C of a common class is defined as:

$$C_C = \frac{\|I_C\|}{n},$$

the commonality C_{VP} of a variation point is defined as:

$$C_{VP} = \frac{\|I_{VP}\|}{n},$$

and the commonality C_{OP} of an optional point is defined as:

$$C_{OP} = \frac{\|I_{OP}\|}{n},$$

where $\|I_C\|$, is the number of instances sharing the common class, $\|I_{VP}\|$ is the number of instances sharing the variation point, $\|I_{OP}\|$ is the number of instances sharing the optional point, and n is the total number of instances in the reference model.

Table 30. Optional Point Commonality, Case Study 3

	Run0	Run1	Run2	Run3	Run4	Avg.
Avg.	0.42	0.41	0.43	0.45	0.39	0.42
Max	0.90	0.70	0.80	0.90	0.80	0.82
Min	0.10	0.10	0.10	0.10	0.10	0.10
Std.	0.20	0.17	0.19	0.19	0.18	0.19

Table 31. Reference Model Commonality, Case Study 3

	Run0	Run1	Run2	Run3	Run4	Avg.
Avg.	0.71	0.70	0.72	0.72	0.70	0.71

According to Definitions 7.6 and 7.7 in Chapter 7, both the common classes and the variation points are shared by all the instances in the reference, making their

commonality ratios C_C and C_{VP} usually 1. The situation is not the same with the optional points. Thus, Table 30 shows the average, the maximum, the minimum, and the standard deviation of the C_{OP} for the optional points in the reference model with 10 generalized instances. An average of 0.42 means that, in average, an optional point in the reference model is shared by 4 to 5 instances

Table 32. Optional Point Commonality, Case Study 1

	Run0	Run1	Run2	Run3	Run4	Avg.
Avg.	0.57	0.57	0.57	0.57	0.57	0.57
Max	0.80	0.80	0.80	0.80	0.80	0.80
Min	0.40	0.40	0.40	0.40	0.40	0.40
Std.	0.13	0.13	0.13	0.13	0.13	0.13

Table 33. Reference Model Commonality, Case Study 1

	Run0	Run1	Run2	Run3	Run4	Avg.
Avg.	0.79	0.79	0.79	0.79	0.79	0.79

The commonality of the reference, shown in

Table 31, is computed as the average of C_C , C_{VP} , and C_{OP} .

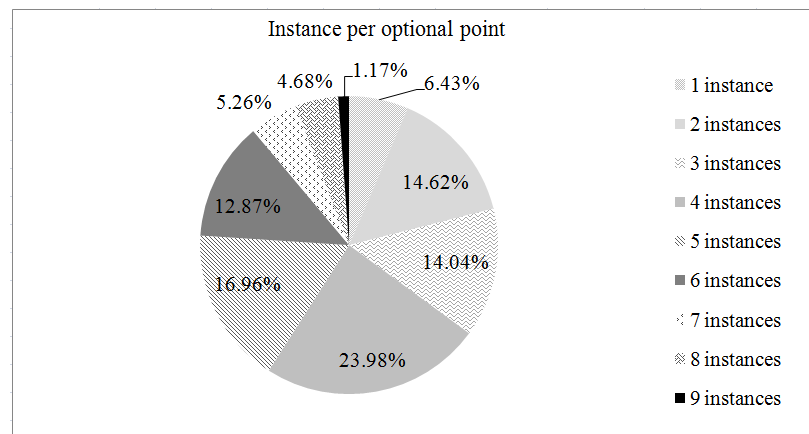


Figure 48. Percentage of Optional Points at Different Commonality Values, CS3

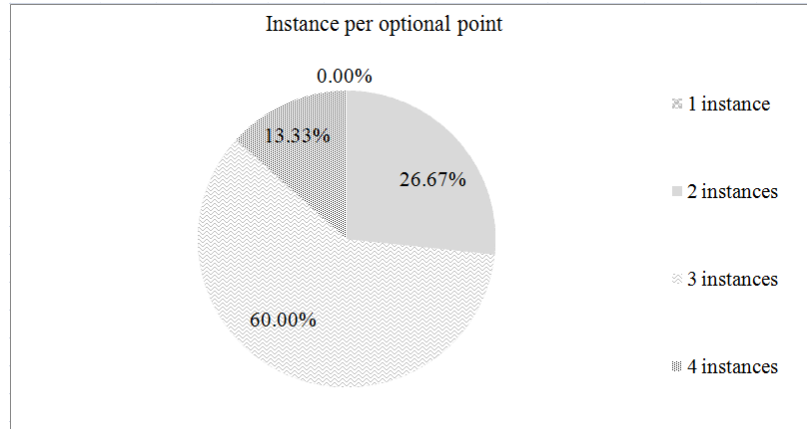


Figure 49. Percentage of Optional Points at Different Commonality Values, Case Study 1

Although Table 30 can tell us the basic statistics about the optional points commonality, it does not tell us for example what is the percentage of optional points having the average, the maximum, and the minimum commonality values, of the overall optional points. Further insight to this is given in Figure 48 (also Figure 49 for Case Study 1), which shows the percentage of optional points at different commonality values. As we can see in the Pie chart, around 41% of the optional points have their C_{OP} value around the average commonality value (i.e. shared by either 4 instances or 5 instances), around 6% of the optional points have their C_{OP} value around the minimum commonality value (i.e. generalize only one instance), and around 6% of the optional points have their C_{OP} value around the maximum commonality value (i.e. shared by either 8 instances or 9 instances). This is an indicator of the reuse opportunity in the reference model, for around 94% of the optional points are shared by more than one instance. Higher commonality ratio was obtained for Case Study 1 see Figure 49 along with Table 32 and

Table 33.

Figure 50 shows the average reuse ratio achieved from the reference model, with regard to the new generalized instance, as compared the reuse ratio achieved from the

best single instance or achieved commonly from all the generalized instances, at different sizes of the reference model. The figure shows three curves representing the reuse ratio achieved from the whole reference model (indicated in the figure as “Multiple”), the reuse ratio achieved from the common part of the reference (indicated in the figure as “All”), and the best reuse ratio achieved from a single instance (indicated in the figure as “Single”). As it is clear from the figure that the reuse ratio offered by the reference model is higher than the best reuse ratio offered from a single instance alone, at different size of the reference model. Additionally, as more instances are generalized to the reference, the reuse ratio offered by the reference increases, where it goes from 80%, when the reference has just two instances, until it reaches 100%, when the number of instances in the reference reaches 8 instances. However, the best reuse ratio achieved from a single instance is between 72% (when the reference has two instances) and 82% (when the reference has 9 instances). The increase in the reuse ratio versus the size of the reference model reflects the motivation and the rationale behind the consolidation process. Moreover, the big difference between the reuse ratio achieve by the reference as a whole and the reuse ratio achieved from the common part in the reference shows clearly the reuse potential involved in the variable part of the reference model. This also justifies the overhead encountered due to managing the variability in the reference and it signifies its importance.

The results in Figure 51 show a situation when one of the models generalized in the reference is a superset of the others, which is the case of the models in Case Study 1. In this case the reuse potential offered by the reference is equivalent to that offered by the superset instance. However, the figure also validates two things. First, it provides a

validation for our merging algorithms. Second, it emphasizes the reuse potential offered by modeling the variability in the reference, as compared to generalizing what is only common. Moreover, though building a reference for multiple releases is beneficial for versioning management, we do not target reuse potential of the reference model to be achieved from generalizing multiple releases.

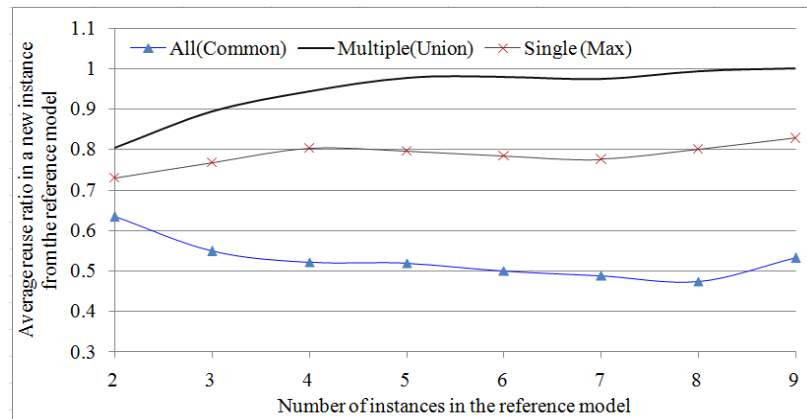


Figure 50. Average Reuse Ratio in a New Instance versus the Size of the Reference Model, Case Study 3.

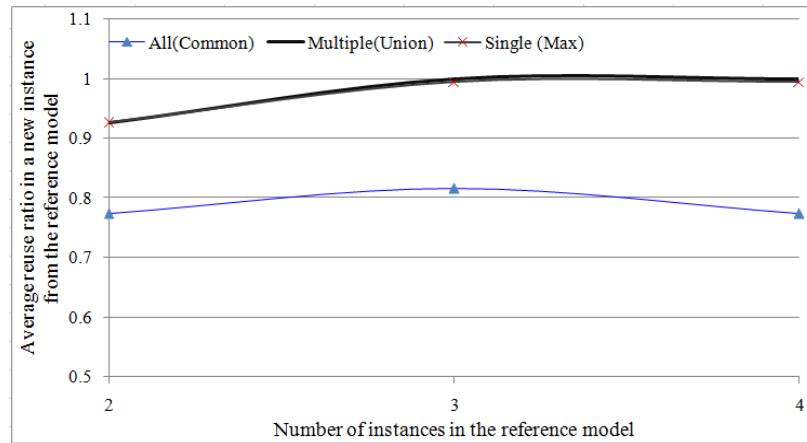


Figure 51. Average Reuse Ratio in a New Instance versus the Size of the Reference Model, Case Study 1.

Table 34. The Standard Deviation of the Reuse Ratio over the Different Runs

	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9
Multiple	0.033	0.031	0.036	0.030	0.027	0.032	0.012	0.000
Single	0.037	0.045	0.035	0.010	0.029	0.042	0.027	0.020
Common	0.050	0.022	0.029	0.012	0.049	0.040	0.034	0.053

Table 34 shows the corresponding standard deviation for five runs averaged in Figure 50. The low standard deviation values indicates that the behavior in the five runs is very close to its average.

Attributes and methods reuse

The improvement in the attributes or methods' reuse as the reference model gets more instances can be measured in terms of the average number of attributes or methods added per class when a new instance is generalized. Figure 52 and Figure 53 respectively show the average number of attributes and methods added to the reference class, due to the generalization of a class of a new instance, against the size of the reference model. As indicated in the figure, this number is decreasing as the reference gets more instances. This indicates the improvement in the reuse offered by the reference at the level of attributes and methods.

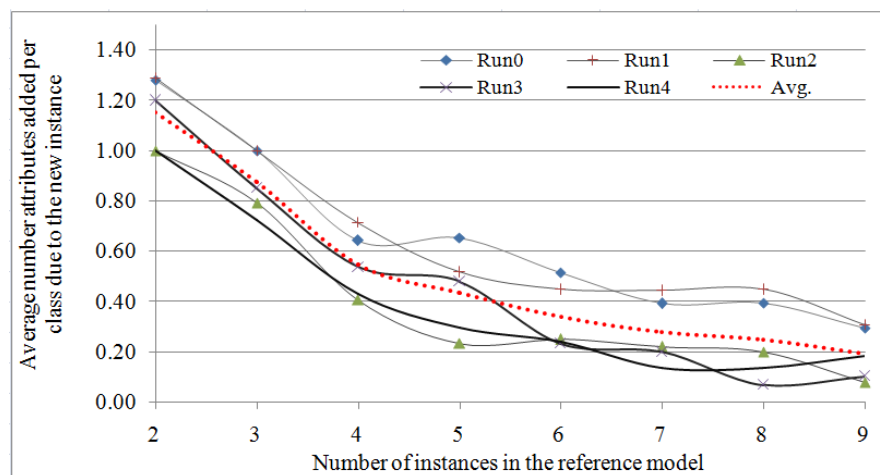


Figure 52. Attributes Added to the Reference Class Per New Instance, CS3.

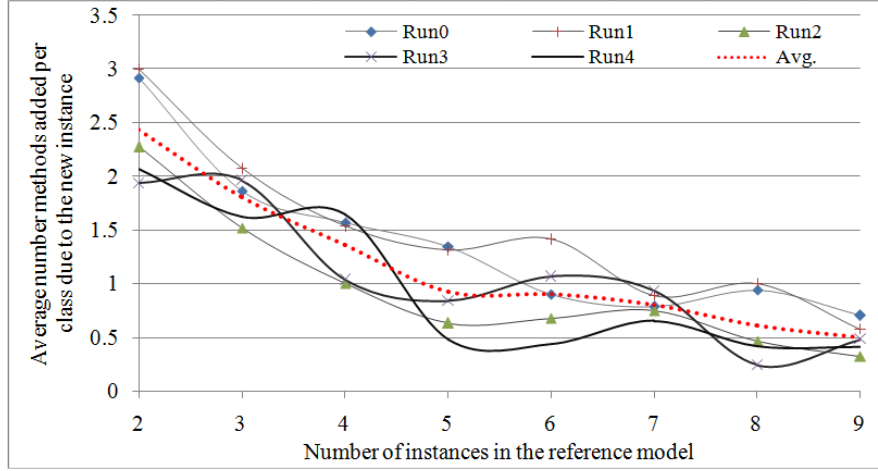


Figure 53. Methods Added to the Reference Class Per New Instance, Case Study 3

Variability overhead

Both variation points and optional points are meant to model the variability among the different models generalized by the reference model. They are additional elements that were not exist in the input models. This means that one can look at them as an overhead. However, this overhead can be justified when these elements represent an abstraction of many instances. For example, in the case of optional point, an optional point can represent a single instance or it can represent $n-1$ instances, where n is the number of instances generalized by the reference. These are two extremes. In the former case the overhead ratio is 1 (i.e. 100%) while in the later case it is $1/(n-1)$. We can define the overhead ratio OH_{OP} of an optional point OP as follows:

$$OH_{OP} = \frac{1}{I_{OP}},$$

where I_{OP} is the number of instances generalized by OP .

Similarly we can define the overhead ratio OH_{VP} of a variation point VP as:

$$OH_{VP} = \frac{1}{I_{VP}},$$

where I_{VP} is the number of instances generalized by VP .

The value of I_{VP} is usually n for all the variation points, whereas the value of I_{OP} varies from an optional point to another optional point. Therefore, the overhead ratio of any variation point is $1/n$ while the overhead ratio of an optional point varies between 1 and $1/(n-1)$.

Table 35 and

Table 36 show the average overhead ratio of optional points at different size for multiple runs, for both Case Study 3 and Case Study 1, respectively. It is clear from the two tables that as n increase the ratio overhead decrease.

Table 35. Optional Points Ratio Overhead at Different Size of the Reference Model, Case Study 3

	Run0	Run1	Run2	Run3	Run4	Avg.
$n=2$	1.00	1.00	1.00	1.00	1.00	1.00
$n=5$	0.64	0.55	0.57	0.55	0.59	0.58
$n=10$	0.31	0.33	0.31	0.29	0.33	0.31

Table 36. Optional Points Ratio Overhead at Different Size of the Reference Model, Case Study 1.

	Run0	Run1	Run2	Run3	Run4	Avg.
$n=2$	1.00	1.00	1.00	1.00	1.00	1.00
$n=4$	0.48	0.58	0.41	0.58	0.48	0.50
$n=5$	0.37	0.37	0.37	0.37	0.37	0.37

Figure 54 shows the number of optional points per instance generalized in the reference so far. If we consider the optional points as an overhead or a cost encountered for modeling the variability in the reference, the figure shows that this overhead is

decreasing as more instances are added to the reference. As shown in the figure, at the beginning, when the reference has few number of instances the number of optional points can go higher as new instances are generalized to the reference. However, as the reference gets more instances this number starts to go down. The former situation can be attributed to two reasons. First, when the reference has few instances, the generalization of an additional instance may results in converting the similarity level of some common classes into optional ones, resulting in an increase in the number of optional points. Second, when the reference has few instances, the diversity among the different instances has not been adequately captured by the reference so far, resulting in a creation of additional optional points to handle the new optional classes introduced by the new instance.

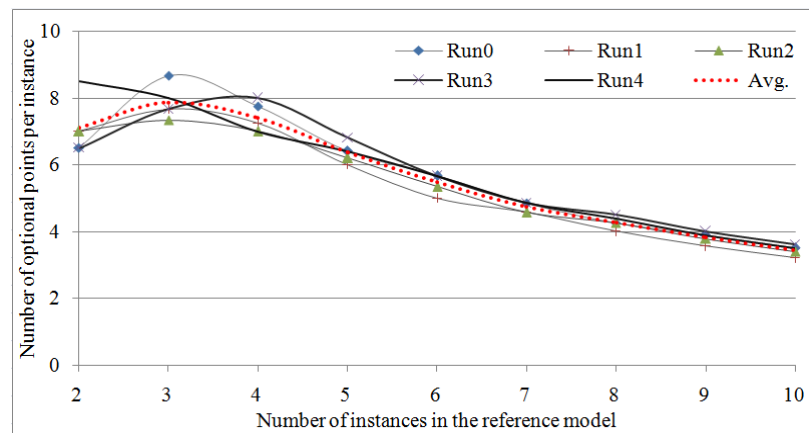


Figure 54. Number of Optional Points Per Instance versus Reference Size, Case Study 3

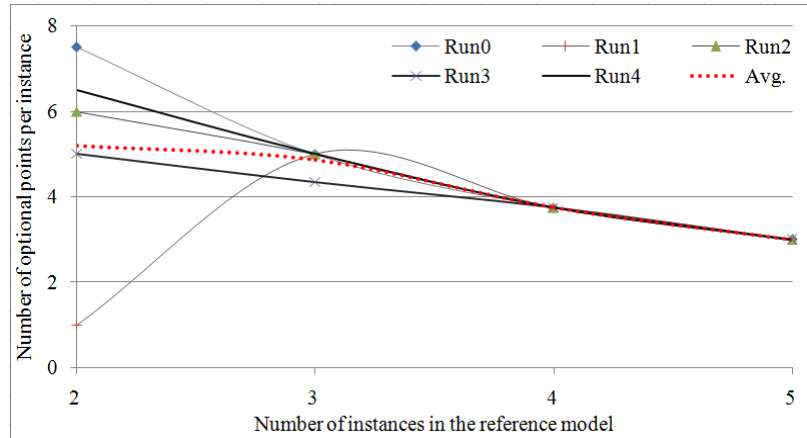


Figure 55. Number of Optional Points Per Instance versus Reference Size, Case Study 1

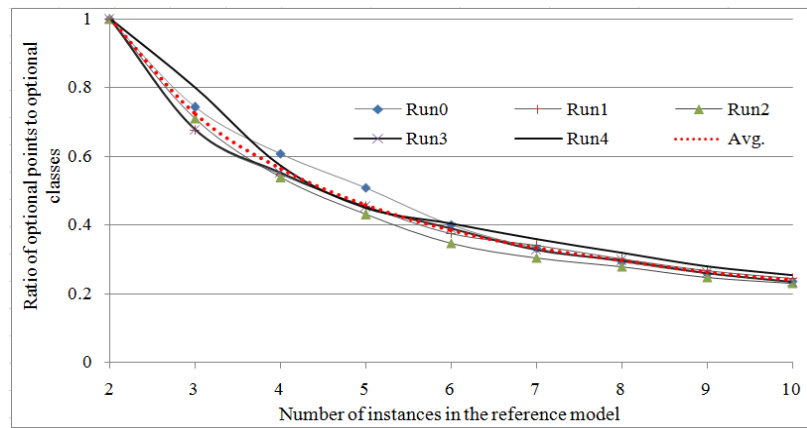


Figure 56. The Ratio of Optional Points to the Optional Classes versus the Reference Model Size, Case Study 3

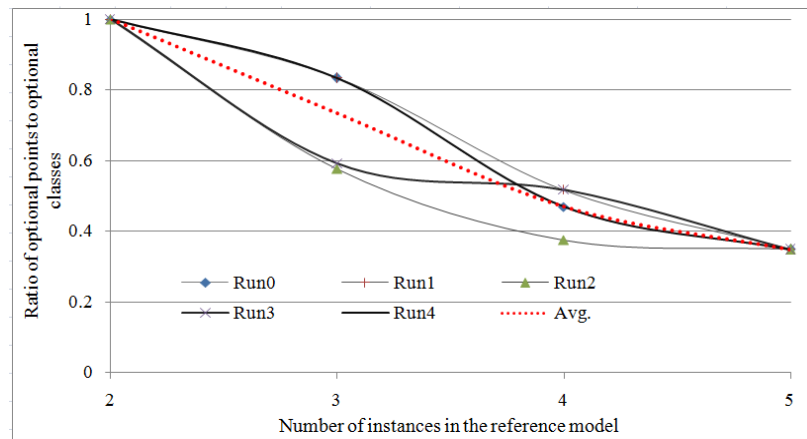


Figure 57. The Ratio of Optional Points to the Optional Classes versus the Reference Model Size, Case Study 1

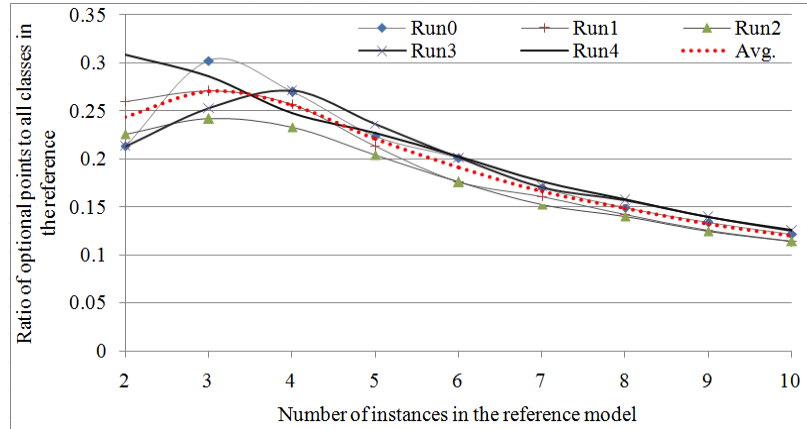


Figure 58. The Ratio of Optional Points to the All Classes versus the Reference Model Size, Case Study 3

Figure 56 and Figure 58 respectively show the ratio of the total number of optional points to the total number of optional classes, and to the total number of all classes generalized to the reference versus the number of instances in the reference. Both figures confirm that as the reference gets more instances the overhead ratio of the optional points gets smaller and smaller.

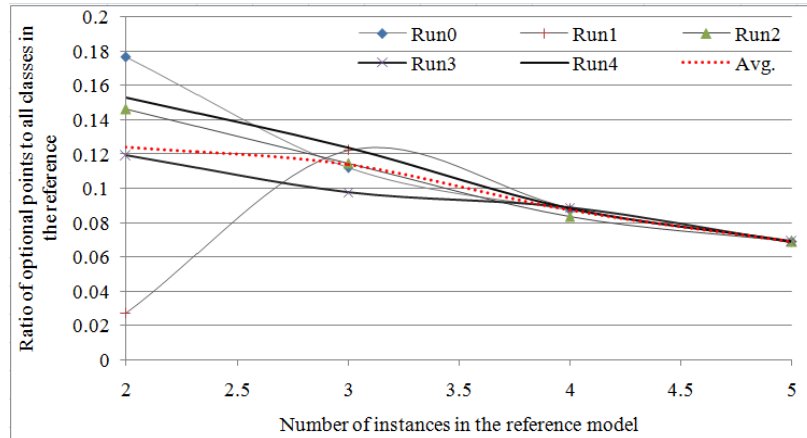


Figure 59. The Ratio of Optional Points to the All Classes versus the Reference Model Size, Case Study 1

To sum up, the results showed that the reference model does offer better reuse ratio than does the best single instance. Modeling variability in the reference, while improving the reuse ratio significantly, has some incurred overhead. However, this overhead is

reasonably acceptable and it gets lower and lower as more instances are generalized into the reference model.

8.9 Threats to Validity

In this section we point out the threats to construct, internal, external, and conclusion validity of our study.

Construct Validity Threats: these types of threats are related to the relationships between theory and the observed findings. In other words, these types of threats can be present when the treatment does not reflect the construct of the cause, or that the outcome does not reflect the construct of the effect [152]. In this regard we can point out a threat related to the similarity metrics used for evaluating the similarity between the elements of the input models. Although the similarity was evaluated based on three different types of similarity information (lexical naming, operations' signature, attributes with their data types, neighborhood information), cooperatively measuring different similarity aspects, and despite the fact that the weights for the different constituents were assigned experimentally, and adding to this the fact that class diagram is considered as the most important artifact in software project development, we still think that other similarity information, such as information from other views, needs to be considered in our future agenda.

Internal Validity Threats: these threats are related to the causal relationship between treatment and outcome [152]. In this regard we can point out two threats. The first threat is concerning the metrics used to evaluate the similarity between the elements of the input models. We do realize that under the confounding effects resulting from the generic attributes or operations, the similarity assessment metrics may not capture the actual similarity between the elements of the matched models and hence may result in a wrong match, which in turn, lead to a wrong consolidation. Nonetheless, the high value of the

similarity threshold as well as the use of combined similarity measures along with the weight calibration procedure is almost obviating this threat. The second threat is concerning the lack of real data. While we used a structured, carefully designed, and clearly described random instances' generator that generates different instances from a real world source, the generated instances may lack properties found in real-world instances.

External Validity Threats: these threats are related to the generalization of the observed findings [152]. In this regard two threats can be pointed out. The first threat is concerning the size of our experimental objects. Although the models we used are of reasonable sizes, we do realize that future investigating of our approach with data of larger sizes is required to confirm its generalization and draw stronger conclusions. The second threat is concerning the generalization of the weight settings experiments' findings. Although we did weight calibration using different case studies, we still think that further validations with different systems are needed to confirm the generalization of our findings and draw stronger conclusions.

Conclusion validity threats: these threats are related to the general relationship between treatment and outcome [152]. In this regard we can point out a threat concerning the scalability of the third stage of the matching. However, dealing with models representing instances within the same domain is expected to have high commonality, and thus the matching of the majority of the elements will be done within the first matching stage in polynomial time, which is also followed by another polynomial time matching stage. Therefore, only few residuals will be investigated in the third stage. This

is actually the gain of the staged matching algorithm, i.e. reducing the time complexity through stage matching.

CHAPTER 9

CONCLUSION AND FUTURE WORK

In this work we proposed a solution framework for generalizing a set of models, representing different applications within a domain or similar domains, into a reference model with the purpose of improving the reuse of early stage artifacts. The rationale underlying our work is that the reuse potential of multiple models can be offered under the complexity of a single model, i.e. the reference model, which unifies the commonality and explicates the variability of the different models it generalizes. The reference model while offering the reuse potential of multiple models, it reduces the complexity of the multiple models into the level of the complexity of a single model.

The proposed solution involves three main activities, model comparison, model matching, and model merging. To tackle the complexity of the problem, we proposed staged matching and merging algorithms.

Our main findings can be summarized as follows.

- Model comparison and similarity.
 - The proposed compound similarity metric for quantifying the degree of similarity between the elements of the input models reported high accuracy under the appropriate weight assignment.
- Matching algorithms.
 - The proposed matching algorithms reported high matching accuracy over the different experimental objects, given accurate similarity values.

- Reference model generalization and reuse.
 - The proposed generalization algorithms capture the commonality and variability at different level of granularity.
 - Reuse can be significantly improved through the reference model as compared to the reuse from a single instance.
 - While maintaining the variability among the different input instances in the reference model involves some overhead, it significantly improves the overall reuse of the reference model, as compared to the reuse offered by the common part of the reference. Additionally this incurred overhead gets minimized as more instances are consolidated into the reference.

9.1 Future work

Directions for future work related to the contribution of this thesis can be outlined in the following.

- Reporting the efficiency and the effectiveness of the proposed approach based on industrial data sets, when available.
- In this work the focus was on a structural view of the software system. Two future directions can be identified here:
 1. The adoption of the proposed framework for the other views.
 2. Improving the proposed framework to consider the multi-views.

References

- [1] H. P. Breivold, S. Larsson, and R. Land, "Migrating Industrial Systems towards Software Product Lines: Experiences and Observations through Case Studies," presented at Proceedings of Euromicro SEAA'08, Washington, DC, USA, 2008.
- [2] Y.-f. Lu and Y.-f. Yin, "A New Constructive Cost Model for Software Testing Project Management," in *The 19th International Conference on Industrial Engineering and Engineering Management*, E. Qi, J. Shen, and R. Dou, Eds.: Springer Berlin Heidelberg, 2013, pp. 545-556.
- [3] M. Nogueira and R. Machado, "Importance of Risk Process in Management Software Projects in Small Companies," in *Advances in Production Management Systems. Innovative and Knowledge-Based Production Management in a Global-Local World*, vol. 439, *IFIP Advances in Information and Communication Technology*, B. Grabot, B. Vallespir, S. Gomes, A. Bouras, and D. Kiritsis, Eds.: Springer Berlin Heidelberg, 2014, pp. 358-365.
- [4] C. W. Krueger, "Software reuse," *ACM Computing Surveys (CSUR)*, vol. 24, pp. 131-183, 1992.
- [5] B. Tekinerdogan and M. Aksit, "A Comparative Analysis of Software Engineering with Mature Engineering Disciplines Using a Problem-Solving Perspective," in *Modern Software Engineering Concepts and Practices: Advanced Approaches*, A. H. D. a. V. Biçer, Ed. New York: IGI Global, 2011, pp. 1-18.
- [6] Gartner Group, "Software Reuse Report," Stanford 1995.
- [7] M. Jha and L. O'Brien, "A comparison of software reuse in software development communities " presented at Software Engineering (MySEC), 2011 5th Malaysian Conference, Johor Bahru, 2011.
- [8] D. C. Rine and N. Nada, "Three empirical studies of a software reuse reference model," *Software: Practice and Experience*, vol. 30, pp. 685-722, 2000.
- [9] W. Frakes, "Systematic software reuse: a paradigm shift," presented at Proceedings of 1994 3rd International Conference on Software Reuse, 1994.
- [10] S. E. de Almeida, "RiDE: The RiSE Process for Domain Engineering," in *Computer Science. RECIFE: Universidade Federal de Pernambuco*, 2007, pp. 278.
- [11] W. B. Frakes and S. Isoda, "Success factors of systematic reuse," *Software, IEEE*, vol. 11, pp. 14 - 19, 1994.
- [12] R. D. Banker, R. J. Kauffman, and D. Zweig, "Repository evaluation of software reuse," *Software Engineering, IEEE Transactions on*, vol. 19, pp. 379-389, 1993.
- [13] M. Morisio, M. Ezran, and C. Tully, "Success and failure factors in software reuse," *Software Engineering, IEEE Transactions on*, vol. 28, pp. 340-357, 2002.
- [14] R. J. Leach, *Software Reuse: Methods, Models, Costs: AfterMath*, 2012.
- [15] M. Petre and D. Damian, "Methodology and culture: drivers of mediocrity in software engineering?," presented at Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014.
- [16] M. Ezran, M. Morisio, and C. Tully, *Practical software reuse*: Springer, 2002.
- [17] R. Salay, S. Wang, and V. Suen, *Managing related models in vehicle control software development*: Springer, 2012.

- [18] J. Rubin and M. Chechik, "From Products to Product Lines Using Model Matching and Refactoring," presented at In: Proc. of SPLC Wrksp, 2010.
- [19] M. Chechik, S. Nejati, and M. Sabetzadeh, "A Relationship-Based Approach to Model Integration," *Innovations in Systems and Software Engineering*, vol. 8, pp. 3-18, 2012.
- [20] W. Tracz, *Software Reuse: Emerging Technology*. New York: IEEE Press, 1988.
- [21] J. L. Cybulski, "Introduction to Software Reuse," University of Melbourne, Melbourne, Australia 1996.
- [22] M. Ahmed, "Towards the Development of Integrated Reuse Environments for UML Artifacts," presented at in ICSEA 2011 : The Sixth International Conference on Software Engineering Advances, 2011.
- [23] J.-M. Goff, Z. Kovacs, N. Baker, R. Brooks, and R. McClatchey, "A Component Based Approach to Scientific Workflow Management." GENEVA, Switzerland, 2001.
- [24] B. Selic, "The Pragmatics of Model-driven Development," *Software, IEEE*, vol. 20, pp. 19-25, 2003.
- [25] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," presented at FOSE '07, Future of Software Engineering, 2007.
- [26] O. Bech, "A Multi-Layer Modelling Environment for Diagram Predicate Framework in Eclipse," Master's Thesis 2011.
- [27] T. Stahl and M. Volter, *Model-Driven Software Development*: Wiley Publishing, 2006.
- [28] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*: Pearson Education, 1994.
- [29] L. Northrop, "Software Product Lines: Reuse That Makes Business Sense," in *Software Engineering Institute (SEI)*, 2007.
- [30] K. Czarnecki, M. Antkiewicz, C. H. P. Kim, S. Lau, and K. Pietroszek, "Model-Driven Software Product Lines," presented at OOPSLA'05, San Diego, California, 2005.
- [31] A. Pleuss, G. Botterweck, D. Dhungana, A. Polzer, and S. Kowalewski, "Model-driven support for product line evolution on feature level," *Journal of Systems and Software*, vol. 85, pp. 2261–2274, 2012.
- [32] H. Gomaa, *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*: Addison-Wesley, 2004.
- [33] J. Rubin and M. Chechik, "Combining Related Products into Product Lines," in *the 15th international conference on Fundamental Approaches to Software Engineering (FASE'12)*, vol. 7212: Lecture Notes in Computer Science, 2012, pp. 285-300.
- [34] J. S. Poulin, "Measuring software reuse: principles, practices, and economic models," 1997.
- [35] M. Smolárová and P. Návrát, "Software reuse: Principles, patterns, prospects," *CIT. Journal of computing and information technology*, vol. 5, pp. 33-49, 1997.
- [36] M. Famelis, R. Salay, and M. Chechik, "Partial models: Towards modeling and reasoning with uncertainty," presented at Software Engineering (ICSE), 2012 34th International Conference on., 2012.

- [37] R. Lutz, D. Wurfel, and S. Diehl, "How Humans merge UML-Models," presented at In Proc. of the International Symposium on Empirical Software Engineering and Measurement, Banff, Alberta, Canada, 2011.
- [38] Y. Xue, "Reengineering legacy software products into software product line based on automatic variability analysis," presented at ICSE '11 Proceedings of the 33rd International Conference on Software Engineering, 2011.
- [39] I. Reinhartz-Berger, "Towards automatization of domain modeling," *Data & Knowledge Engineering*, vol. 69, pp. 491–515, 2010.
- [40] T. Han, S. Purao, and V. C. Storey, "Generating large-scale repositories of reusable artifacts for conceptual design of information systems," *Decision Support Systems*, vol. 45, pp. 665–680, 2008.
- [41] C. Li, M. Reichert, and A. Wombacher, "Mining business process variants: Challenges, scenarios, algorithms," *Data & Knowledge Engineering*, vol. 70, pp. 409–434, 2011.
- [42] M. Sabetzadeh and S. Easterbrook, "View merging in the presence of incompleteness and inconsistency," vol. 11, pp. 174–193, 2006.
- [43] M. Alanen and I. Porres, "Difference and Union of Models," in *UML 2003 - The Unified Modeling Language*, vol. 2863 LNCS, 2003, pp. 2–17.
- [44] P. Roques, *UML in practice: the art of modeling software systems demonstrated through worked examples and solutions*: Wiley, 2006.
- [45] K. Robles, A. Fraga, J. Morato, and J. Llorens, "Towards an ontology-based retrieval of UML Class Diagrams," *Information and Software Technology*, vol. 54, pp. 72–86, 2012.
- [46] S. Apel, F. Janda, S. Trujillo, and C. Kästner, "Model superimposition in software product lines," in *Theory and Practice of Model Transformations*: Springer, 2009, pp. 4–19.
- [47] P. Mohagheghi, V. Dehlen, and T. Neple, "Definitions and approaches to model quality in model-based software development—A review of literature," *Information and Software Technology*, vol. 51, pp. 1646–1669, 2009.
- [48] T. Levendovszky, B. Rumpe, B. Schätz, and J. Sprinkle, "9 Model Evolution and Management," in *Model-Based Engineering of Embedded Real-Time Systems*: Springer, 2010, pp. 241–270.
- [49] T. Clark and P.-A. Muller, "Exploiting model driven technology: a tale of two startups," *Software & Systems Modeling*, vol. 11, pp. 481–493, 2012.
- [50] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave, "Matching and merging of Statechart specifications," in *the 29th International Conference on Software Engineering (ICSE'07)* Minneapolis, MN, USA: IEEE Computer Society, 2007, pp. 54–64.
- [51] G. Brunet, M. Chechik, S. Easterbrook, N. Nejati, N. Niu, and M. Sabetzadeh, "A manifesto for model merging," presented at Workshop on Global Integrated Model Management (GaMMA'06) co-located with ICSE'06,, 2006.
- [52] Y. Lin, J. Zhang, and J. Gray, "Model comparison: A key challenge for transformation testing and version control in model driven software development," presented at OOPSLA Workshop on Best Practices for Model-Driven Software Development, 2004.

- [53] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis Lectures on Software Engineering*, vol. 1, pp. 1-182, 2012.
- [54] S. Melnik, H. Garcia-Molina, and E. Rahm, "Similarity flooding: A versatile graph matching algorithm and its application to schema matching," presented at Proc. 18th ICDE Conf.(Best Student Paper award), 2002.
- [55] R. Pottinger and P. Bernstein, "Merging models based on given correspondences," presented at In Proceedings of 29th International Conference on Very Large Data Bases, 2003.
- [56] D. S. Kolovos, D. Di Ruscio, A. Pierantonio, and R. F. Paige, "Different models for model matching: An analysis of approaches to support model differencing," in *ICSE Workshop on Comparison and Versioning of Software Models*: IEEE, 2009, pp. 1-6.
- [57] M. Amstel, M. Brand, Z. Protic, and T. Verhoeff, "Model-Driven Software Engineering," in *Automation in Warehouse Development*, R. Hamberg and J. Verriet, Eds.: Informs 5521 Research Park DR, Suite 200, Catonsville, MD 21228 USA, 2012, pp. 53.
- [58] D. S. Kolovos, R. F. Paige, and F. A. Polack, "Merging models with the epsilon merging language (EML)," in *Model Driven Engineering Languages and Systems*: Springer, 2006, pp. 215-229.
- [59] Y. Lin, J. Gray, and F. Jouault, "DSMDiff: a differentiation tool for domain-specific models," *European Journal of Information Systems*, vol. 16, pp. 349-361, 2007.
- [60] U. Kelter, J. Wehren, and J. Niere, "A Generic Difference Algorithm for UML Models," presented at In Software Engineering, Essen, 2005.
- [61] Z. Xing and E. Stroulia, "UMLDiff: An algorithm for object-oriented design differencing," in *the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE '05)* ACM, New York, NY, USA, 2005, pp. 54-65.
- [62] L. Montrieux, M. Wermelinger, and Y. Yu, "Challenges in model-based evolution and merging of access control policies," presented at Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution, 2011.
- [63] <http://www.uml.org/>.
- [64] <http://www.omg.org/>.
- [65] S. Roh, K. Kim, and T. Jeon, "Architecture modeling language based on UML2.0," presented at Software Engineering Conference, 2004. 11th Asia-Pacific, 2004.
- [66] <http://www.omg.org/mof/>.
- [67] <http://www.omg.org/spec/XMI/>.
- [68] T. J. Grose, G. C. Doney, and S. A. Brodsky, *Mastering Xmi: Java Programming with Xmi, XML and UML*, vol. 21: John Wiley & Sons, 2002.
- [69] J. Kovse and T. Härder, "Generic XMI-based UML model transformations," in *Object-oriented information systems*: Springer, 2002, pp. 192-198.
- [70] www.altova.com.
- [71] <http://www.w3.org/>.
- [72] P. O'Leary and I. Richardson, "Process reference model construction: implementing an evolutionary multi-method research approach," *Software, IET*, vol. 6, pp. 423-430, 2012.

- [73] I. Sommerville, "Software engineering—9th ed. p. cm," McGraw-Hill Companies Inc., New York, 2011.
- [74] M. Rosemann and W. M. P. van der Aalst, "A Configurable Reference Modeling Language," *Information Systems*, vol. 23, pp. 1-23, 2007.
- [75] R. K. Ahuja, J. B. Orlin, and A. Tiwari, "A greedy genetic algorithm for the quadratic assignment problem," *Computers & Operations Research*, vol. 27, pp. 917-934, 2000.
- [76] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Artificial Intelligence Review*, vol. 13, pp. 129-170, 1999.
- [77] S. Roy, "Genetic algorithm based approach to solve travelling salesman problem with one point crossover operator," *International Journal Of Computers & Technology*, vol. 10, pp. 1393-1400, 2013.
- [78] H. O. Salami and M. A. Ahmed, "A Framework for Class Diagram Retrieval Using Genetic Algorithm," in *the 24th Int. Conf. on Software Engineering and Knowledge Engineering SEKE'12*, 2012, pp. 737-740.
- [79] A. D. Cross, R. C. Wilson, and E. R. Hancock, "Inexact graph matching using genetic search," *Pattern Recognition*, vol. 30, pp. 953-970, 1997.
- [80] J. Brownlee, *Clever algorithms: nature-inspired programming recipes*: Jason Brownlee, 2011.
- [81] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*: U Michigan Press, 1975.
- [82] X. Yan, W. Luo, W. Li, W. Chen, C. Zhang, and H. Liu, "An Improved Genetic Algorithm and Its Application in Classification," *International Journal of Computer Science Issues (IJCSI)*, vol. 10, 2013.
- [83] S. Kirkpatrick and M. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, pp. 671-680, 1983.
- [84] Z. Michalewicz and D. B. Fogel, *How to solve it: Modern Heuristics*. New York:: Springer, 2000.
- [85] G. Xiutang and Z. Kai, "Simulated annealing algorithm for detecting graph isomorphism," *Journal of Systems Engineering and Electronics*, vol. 19, pp. 1047-1052, 2008.
- [86] M. Malek, M. Guruswamy, M. Pandya, and H. Owens, "Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem," *Annals of Operations Research*, vol. 21, pp. 59-84, 1989.
- [87] S. Fidanova, "Simulated annealing for grid scheduling problem," presented at Modern Computing, 2006. JVA'06. IEEE John Vincent Atanasoff 2006 International Symposium on, 2006.
- [88] J. Bergey, P. Clements, S. Cohen, P. Donohoe, L. Jones, B. Krut, L. Northrop, S. Tilley, D. Smith, and J. Withey, "DoD Product Line Practice," Software Engineering Institute, Carnegie Mellon University, Pittsburgh 1998.
- [89] H. P. Breivold, S. Larsson, and R. Land, "Migrating Industrial Systems towards Software Product Lines: Experiences and Observations through Case Studies," presented at In Euromicro SEAA'08 Proceedings, Washington, DC, USA, 2008.

- [90] M. Acher, P. Collet, P. Lahire, and R. France, "Managing Multiple Software Product Lines Using Merging Techniques," University of Nice Sophia Antipolis, I3S CNRS, Sophia Antipolis, France 2010.
- [91] I. Reinhartz-Berger, P. Soffer, and A. Sturm, "A Domain Engineering Approach to Specifying and Applying Reference Models," presented at Workshop Enterprise Modelling and Information Systems Architectures, 2005.
- [92] M. Sinnema and S. Deelstra, "Classifying variability modeling techniques," *Information and Software Technology*, vol. 49, pp. 717–739, 2007.
- [93] L. Chen, M. Babar, and N. Ali, "Variability management in software product lines: a systematic review," presented at SPLC '09 Proceedings of the 13th International Software Product Line Conference, 2009.
- [94] S. D. Kim and H. G. Min, "DREAM: a practical product line engineering using model driven architecture," presented at ICITA '05 Proceedings of the Third International Conference on Information Technology and Applications (ICITA'05), 2005.
- [95] C. W. Krueger and B. Bakal, "Leveraging the Model Driven Development and Software Product Line Engineering Synergy for Success," IBM, White paper 5 May 2008 2008.
- [96] I. Groher, H. Papajewski, and M. Voelter, "Integrating model-driven development and software product line engineering," presented at In Eclipse Modeling Symposium, Ludwigsburg, Germany, 2007.
- [97] I. Groher and M. Voelter, "Expressing feature-based variability in structural models," in *Workshop on Managing Variability for Software Product Lines*, 2007.
- [98] S. Barrett, P. Chalin, and G. Butler, "Model merging falls short of software engineering needs," presented at 2nd Workshop on Model-Driven Software Evolution, MoDSE '08, 2008.
- [99] P. Bernstein, H. Halevy, and R. Pottinger, "A vision for management of complex models," *ACM Sigmod Record*, vol. 29, pp. 55-63, 2000.
- [100] K. Kim, H. Kim, and W. Kim, "Building Software Product Line from the Legacy Systems: Experience in the Digital Audio and Video Domain," presented at in Proceedings of the 11th International Software Product Line Conference (SPLC'07), 2007.
- [101] K. Bogdanov and N. Walkinshaw, "Computing the structural difference between state-based models.," in *Reverse Engineering, 2009. WCRE'09. 16th Working Conference on:* IEEE, 2009, pp. 177-186.
- [102] A. M. Geoffrion, "Integrated Modeling Systems," *Computer Science in Economics and Management*, vol. 2, pp. 3-15, 1989.
- [103] T. Ziadi, L. Helouet, and J.-M. Jezequel, "Towards a UML Profile for Software Product Lines," presented at in Product Family Engineering (PFE), Siena, Italy, 2003.
- [104] E. A. Aydin, H. Oguztuzun, A. H. Dogru, and A. S. Karatas, "Merging multi-view feature models by local rules," presented at 9th International Conference on Software, 2011.
- [105] J. Bayer, J.-F. Girard, M. Wurthner, J.-M. DeBaud, and M. Apel, "Transitioning Legacy Assets to a Product Line Architecture," presented at 7th ACM SIGSOFT

- International Symposium on Foundations of Software Engineering (FSE'99), Toulouse, 1999.
- [106] S. Ferber, J. Haag, and J. Savolainen, "Feature Interaction and Dependencies: Modeling Features for Reengineering a Legacy Product Line," presented at in Proceedings of the Second International Conference on Software Product Lines (SPLC'02), 2002.
 - [107] F. Fleurey, B. Baudry, R. France, and G. S., "A Generic Approach for Automatic Model Composition," presented at Models in Software Engineering, 2008.
 - [108] N. Noda and T. Kishi, "Aspect-Oriented Modeling for Variability Management," presented at In SPLC '08: Proceedings of the 2008 12th International Software Product Line Conference, Washington, DC, USA, 2008.
 - [109] B. Morin, J. Klein, O. Barais, and J. M. Jezequel, "A generic weaver for supporting product lines," presented at In: Early Aspects Workshop at ICSE, Leipzig, Germany 2008.
 - [110] P. Selonen and M. Kettunen, "Metamodel-Based Inference of Inter-Model Correspondence," presented at The 11th European Conference on Software Maintenance and Reengineering, 2007. CSMR '07. , 2007.
 - [111] U. Kelter and M. Schmidt, "Comparing State Machines," presented at in Proceedings of the 2008 International Workshop on Comparison and Versioning of Software Models (CVSM'08), 2008.
 - [112] A. Mehra, J. Grundy, and J. Hosking, "A generic approach to supporting diagram differencing and merging for collaborative design," presented at In ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, 2005.
 - [113] A. Hallerbach, T. Bauer, and M. Reichert, "Capturing Variability in Business Process Models: The Provop Approach," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 22, pp. 519-546, 2010.
 - [114] M. Voelter and I. Groher, "Product Line Implementation using Aspect-Oriented and Model-Driven Software Development," presented at in SPLC, Washington, DC, USA, 2007.
 - [115] M. Acher, P. Collet, and R. France, "Composing Feature Models," presented at 2nd International Conference on Software Language Engineering (SLE'09), 2009.
 - [116] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *The VLDB Journal*, vol. 10, pp. 334-350, 2001.
 - [117] N. Bouassida and H. Ben-Abdallah, "Pattern and spoiled pattern detection through an information retrieval approach," *Journal of Emerging Technologies in Web Intelligence*, vol. 2, pp. 167-175, 2010.
 - [118] C. Bouhours, H. Leblanc, C. Percebois, and T. Millan, "Detection of Generic Micro-architectures on Models," presented at The Second International Conferences on Pervasive Patterns and Applications (PATTERNS 2010), 2010.
 - [119] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis, "Design Pattern Detection Using Similarity Scoring," *IEEE Transactions on Software Engineering*, vol. 32, pp. 896-909, 2006.
 - [120] T. Pedersen, S. Patwardhan, and J. Michelizzi, "WordNet:: Similarity: measuring the relatedness of concepts," in *the 19th National Conference on Artificial*

- Intelligence*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2004, pp. 38-41
- [121] D. Parnas, "Designing Software for Ease of Extension and Contraction," *IEEE Transactions on Software Engineering*, vol. 5, pp. 128–138, 1979.
 - [122] H. Bunke, P. Foggia, C. Guidobaldi, C. Sansone, and M. Vento, "A Comparison of Algorithms for Maximum Common Subgraph on Randomly Connected Graphs," *Lecture Notes in Computer Science* vol. 2396, pp. 123-132 2002.
 - [123] D. S. Kolovos, "Establishing correspondences between models with the epsilon comparison language," in *Model Driven Architecture-Foundations and Applications*: Springer, 2009, pp. 146-157.
 - [124] R. A. Rufai, "New structural similarity metrics for UML models," in *Computer Science*. Dhahran: King Fahd University of Petroleum & Minerals, 2003.
 - [125] M. A.-R. Al-Khiaty and M. Ahmed, "Similarity Assessment of UML Class Diagrams using a Greedy Algorithm," in *The 18th International Computer Science and Engineering Conference (ICSEC2014)*. Khon Kaen, Thailand: IEEE, 2014, pp. 243-248.
 - [126] M. Keshavarz and Y.-H. Lee, "Ontology matching by using ConceptNet," in *the Asia Pacific Industrial Engineering & Management Systems*, 2012, pp. 1917-1925.
 - [127] P. Resnik, "Using information Content to evaluate semantic similarity in a taxonomy," in *the 14th international joint conference on Artificial intelligence*, vol. 1, 1995, pp. 448-453.
 - [128] Z. Wu and M. Palmer, "Verb semantics and lexical selection," in *the 32nd Annual Meeting of the Association for Computational Linguistics*, 1994, pp. 133-138.
 - [129] V. Cross and X. Hu, "Using semantic similarity in ontology alignment," in *the Sixth International Workshop on Ontology Matching (collocated with ISWC-2011)*, 2011, pp. 61-72.
 - [130] S. Patwardhan, "Incorporating dictionary and corpus information into a context vector measure of semantic relatedness," University of Minnesota, 2003.
 - [131] A. Budanitsky and G. Hirst, "Evaluating wordnet-based measures of lexical semantic relatedness," *Computational Linguistics*, vol. 32, pp. 13-47, 2006.
 - [132] M. J. Sussna, "Text Retrieval Using Inference in Semantic Metanetworks." San Diego: University of California, 1997.
 - [133] P. J. d. S. Gomes, "A case-based approach to software design," 2004.
 - [134] E. Ozcan and C. K. Mohan, "Partial shape matching using genetic algorithms," *Pattern Recognition Letters*, vol. 18, pp. 987-992, 1997.
 - [135] S. Sivanandam and S. Deepa, *Introduction to genetic algorithms*: Springer, 2007.
 - [136] D. E. Goldberg and R. Lingle Jr, "AllelesLociand the Traveling Salesman Problem," presented at Proceedings of the 1st international conference on genetic algorithms, 1985.
 - [137] L. Davis, "Applying adaptive algorithms to epistatic domains," presented at IJCAI, 1985.
 - [138] K. Deep and H. M. Adane, "New Variations of Order Crossover for Travelling Salesman Problem," *International Journal of Combinatorial Optimization Problems and Informatics*, vol. 2, pp. 2-13, 2010.

- [139] M. Aurnhammer and K. Tonnie, "A genetic algorithm for automated horizon correlation across faults in seismic images," *Evolutionary Computation, IEEE Transactions on*, vol. 9, pp. 201-210, 2005.
- [140] E. Elbeltagi, T. Hegazy, and D. Grierson, "Comparison among five evolutionary-based optimization algorithms," *Advanced engineering informatics*, vol. 19, pp. 43-53, 2005.
- [141] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 24, pp. 656-667, 1994.
- [142] S. Kirkpatrick, "Optimization by simulated annealing: Quantitative studies," *Journal of statistical physics*, vol. 34, pp. 975-986, 1984.
- [143] D. Henderson, S. H. Jacobson, and A. W. Johnson, "The theory and practice of simulated annealing," in *Handbook of metaheuristics*: Springer, 2003, pp. 287-319.
- [144] E. B. Fernandez and X. Yuan, "Semantic analysis patterns," in *Conceptual Modeling—ER 2000*: Springer, 2000, pp. 183-195.
- [145] W.-S. Li and C. Clifton, "SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks," *Data & Knowledge Engineering*, vol. 33, pp. 49-84, 2000.
- [146] H.-H. Do, S. Melnik, and E. Rahm, "Comparison of schema matching evaluations," in *Web, Web-Services, and Database Systems*: Springer, 2003, pp. 221-237.
- [147] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg, "Adaptive name matching in information integration," *IEEE Intelligent Systems*, vol. 18, pp. 16-23, 2003.
- [148] <https://code.google.com/p/ws4j>.
- [149] J. Eno and C. W. Thompson, "Generating Synthetic Data to Match Data Mining Patterns," *Internet Computing, IEEE*, vol. 12, pp. 78-82, 2008.
- [150] R. Heradio-Gil, D. Fernandez-Amoros, J. A. Cerrada, and C. Cerrada, "Supporting commonality-based analysis of software product lines," *IET software*, vol. 5, pp. 496-509, 2011.
- [151] P. C. Clements, J. D. McGregor, and S. G. Cohen, "The structured intuitive model for product line economics (SIMPLE)," DTIC Document 2005.
- [152] C. Wohlin, P. Runeson, M. Host, C. Ohlsson, B. Regnell, and A. Wesslén, "Experimentation in Software Engineering: an Introduction," 2000.

Vitae

Name : MOJEEB AL-RHMAN AHMED SALEH AL-KHIATY |
Nationality : Yemeni |
Date of Birth : 01-01-1976,|
Email : alkhiaty@gmail.com|
Address : 14, Alwahdah Street, Sana'a, Yemen. |

Academic Background : Al-Khiaty received his Bachelor of Science (BS) degree with honors in Mathematics/Computer from Sana'a University, Yemen, in June 1999. He obtained his Master of Science (MS) degree in Computer Science from King Fahd University of Petroleum and Minerals (KFUPM) in June 2009. Prior to attending KFUPM, he worked as a full time lecturer from September 2000 to Jun 2004 in Sana'a University (Mathematics Department). He joined KFUPM as a full time student to pursue the PhD degree in September 2009. He received his PhD degree in Computer Science and Engineering from KFUPM in February 2015.

During the study of his PhD, Al-Khiaty has published several articles in high quality journals. He has also attended and delivered research papers to good quality international conferences. Additionally, Al-Khiaty has participated in several scientific conferences organized for higher education students in KSA. His research interests include Software Engineering and Soft Computing.